

27th Annual European Symposium on Algorithms

ESA 2019, September 9–11, 2019, Munich/Garching, Germany

Edited by

Michael A. Bender

Ola Svensson

Grzegorz Herman



Editors

Michael A. Bender

Stony Brook University, NY, USA
bender@cs.stonybrook.edu

Ola Svensson 

EPFL, Lausanne, Switzerland
ola.svensson@epfl.ch

Grzegorz Herman 

Jagiellonian University, Kraków, Poland
gherman@tcs.uj.edu.pl

ACM Classification 2012

Applied computing → Transportation; Computing methodologies → Algebraic algorithms; Hardware → External storage; Human-centered computing → Graph drawings; Information systems → Data structures; Mathematics of computing → Algebraic topology; Mathematics of computing → Approximation algorithms; Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Combinatorial optimization; Mathematics of computing → Graph algorithms ; Mathematics of computing → Graph algorithms; Mathematics of computing → Graphs and surfaces; Mathematics of computing → Graph theory; Mathematics of computing → Graph Theory; Mathematics of computing → Integer programming; Mathematics of computing → Linear programming; Mathematics of computing → Network flows; Mathematics of computing → Network optimization; Mathematics of computing → Paths and connectivity problems; Mathematics of computing → Probabilistic algorithms; Mathematics of computing → Probability and statistics; Mathematics of computing → Semidefinite programming; Mathematics of computing → Spectra of graphs; Networks → Network algorithms; Networks → Routing protocols; Theory of computation → Algorithm design techniques; Theory of computation → Algorithmic game theory and mechanism design; Theory of computation → Algorithmic mechanism design; Theory of computation → Approximation algorithms analysis; Theory of computation → Computational complexity and cryptography; Theory of computation → Computational geometry; Theory of computation → Convex optimization; Theory of computation → Data compression; Theory of computation → Data structures design and analysis; Theory of computation → Design and analysis of algorithms; Theory of computation → Dynamic graph algorithms; Theory of computation → Exact and approximate computation of equilibria; Theory of computation → Facility location and clustering; Theory of computation → Fixed parameter tractability; Theory of computation → Graph algorithms analysis; Theory of computation → Interactive proof systems; Theory of computation → Linear programming; Theory of computation → Lower bounds and information complexity; Theory of computation → Massively parallel algorithms; Theory of computation → Mathematical optimization; Theory of computation → Nearest neighbor algorithms; Theory of computation → Network flows; Theory of computation → Numeric approximation algorithms; Theory of computation → Online algorithms; Theory of computation → Packing and covering problems; Theory of computation → Parallel algorithms; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Pattern matching; Theory of computation → Problems, reductions and completeness; Theory of computation → Quantum computation theory; Theory of computation → Quantum query complexity; Theory of computation → Routing and network design problems; Theory of computation → Scheduling algorithms; Theory of computation → Shortest paths; Theory of computation → Sketching and sampling; Theory of computation → Sparsification and spanners; Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Submodular optimization and polymatroids; Theory of computation → Unsupervised learning and clustering; Theory of computation → W hierarchy

ISBN 978-3-95977-124-5*Published online and open access by*

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at <https://www.dagstuhl.de/dagpub/978-3-95977-124-5>.

Publication date

September, 2019

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 <https://portal.dnb.de>.

License

This work is licensed under a Creative Commons Attribution 3.0 Unported license (CC-BY 3.0):

<https://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.2019.0

ISBN 978-3-95977-124-5

ISSN 1868-8969

<https://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)
- Christel Baier (TU Dresden)
- Mikolaj Bojanczyk (University of Warsaw)
- Roberto Di Cosmo (INRIA and University Paris Diderot)
- Javier Esparza (TU München)
- Meena Mahajan (Institute of Mathematical Sciences)
- Dieter van Melkebeek (University of Wisconsin-Madison)
- Anca Muscholl (University Bordeaux)
- Luke Ong (University of Oxford)
- Catuscia Palamidessi (INRIA)
- Thomas Schwentick (TU Dortmund)
- Raimund Seidel (Saarland University and Schloss Dagstuhl – Leibniz-Zentrum für Informatik)

ISSN 1868-8969

<https://www.dagstuhl.de/lipics>

■ Contents

Preface	
<i>Michael A. Bender, Ola Svensson, and Grzegorz Herman</i>	0:xi
Constant-Factor FPT Approximation for Capacitated k -Median	
<i>Marek Adamczyk, Jarosław Byrka, Jan Marcinkowski, Syed M. Meesum, and Michał Włodarczyk</i>	1:1–1:14
Fragile Complexity of Comparison-Based Algorithms	
<i>Peyman Afshani, Rolf Fagerberg, David Hammer, Riko Jacob, Irina Kostitsyna, Ulrich Meyer, Manuel Penschuck, and Nodari Sitchinava</i>	2:1–2:19
Universal Reconfiguration of Facet-Connected Modular Robots by Pivots: The $O(1)$ Musketeers	
<i>Hugo A. Akitaya, Esther M. Arkin, Mirela Damian, Erik D. Demaine, Vida Dujmović, Robin Flatland, Matias Korman, Belen Palop, Irene Parada, André van Renssen, and Vera Sacristán</i>	3:1–3:14
Constructing Light Spanners Deterministically in Near-Linear Time	
<i>Stephen Alstrup, Søren Dahlgaard, Arnold Filtser, Morten Stöckel, and Christian Wulff-Nilsen</i>	4:1–4:15
Repetition Detection in a Dynamic String	
<i>Amihood Amir, Itai Boneh, Panagiotis Charalampopoulos, and Eitan Konradovsky</i>	5:1–5:18
Longest Common Substring Made Fully Dynamic	
<i>Amihood Amir, Panagiotis Charalampopoulos, Solon P. Pissis, and Jakub Radoszewski</i>	6:1–6:17
Bilu-Linial Stability, Certified Algorithms and the Independent Set Problem	
<i>Haris Angelidakis, Pranjal Awasthi, Avrim Blum, Vaggos Chatziafratis, and Chen Dan</i>	7:1–7:16
On the Complexity of Anchored Rectangle Packing	
<i>Antonios Antoniadis, Felix Biermeier, Andrés Cristi, Christoph Damerius, Ruben Hoeksma, Dominik Kaaser, Peter Kling, and Lukas Nölke</i>	8:1–8:14
Quantum Walk Sampling by Growing Seed Sets	
<i>Simon Apers</i>	9:1–9:12
PUFFINN: Parameterless and Universally Fast Finding of Nearest Neighbors	
<i>Martin Aumüller, Tobias Christiani, Rasmus Pagh, and Michael Vesterli</i>	10:1–10:16
Online Multistage Subset Maximization Problems	
<i>Evrpidis Bampis, Bruno Escoffier, Kevin Schewior, and Alexandre Teiller</i>	11:1–11:14
A Constant Approximation for Colorful k -Center	
<i>Sayan Bandyapadhyay, Tanmay Inamdar, Shreyas Pai, and Kasturi Varadarajan</i> ..	12:1–12:14
Parametrized Complexity of Expansion Height	
<i>Ulrich Bauer, Abhishek Rathod, and Jonathan Spreer</i>	13:1–13:15



UnLimited TRAnsfers for Multi-Modal Route Planning: An Efficient Solution <i>Moritz Baum, Valentin Buchhold, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf</i>	14:1–14:16
Streaming and Massively Parallel Algorithms for Edge Coloring <i>Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Marina Knittel, and Hamed Saleh</i>	15:1–15:14
Quantum Algorithms for Classical Probability Distributions <i>Aleksandrs Belovs</i>	16:1–16:11
More Applications of the d -Neighbor Equivalence: Connectivity and Acyclicity Constraints <i>Benjamin Bergougnoux and Mamadou Moustapha Kanté</i>	17:1–17:14
Online Bin Covering with Limited Migration <i>Sebastian Berndt, Leah Epstein, Klaus Jansen, Asaf Levin, Marten Maack, and Lars Rohwedder</i>	18:1–18:14
Computing k -Modal Embeddings of Planar Digraphs <i>Juan José Besa, Giordano Da Lozzo, and Michael T. Goodrich</i>	19:1–19:16
Cost Sharing over Combinatorial Domains: Complement-Free Cost Functions and Beyond <i>Georgios Birmpas, Evangelos Markakis, and Guido Schäfer</i>	20:1–20:17
Efficiently Generating Geometric Inhomogeneous and Hyperbolic Random Graphs <i>Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, Ulrich Meyer, Manuel Penschuck, and Christopher Weyand</i>	21:1–21:14
Randomized Incremental Construction of Delaunay Triangulations of Nice Point Sets <i>Jean-Daniel Boissonnat, Olivier Devillers, Kunal Dutta, and Marc Glisse</i>	22:1–22:13
Fine-Grained Complexity of k -OPT in Bounded-Degree Graphs for Solving TSP <i>Édouard Bonnet, Yoichi Iwata, Bart M. P. Jansen, and Łukasz Kowalik</i>	23:1–23:14
Linear Transformations Between Colorings in Chordal Graphs <i>Nicolas Bousquet and Valentin Bartier</i>	24:1–24:15
Patching Colors with Tensors <i>Cornelius Brand</i>	25:1–25:16
On Geometric Set Cover for Orthants <i>Karl Bringmann, Sándor Kisfaludi-Bak, Michał Pilipczuk, and Erik Jan van Leeuwen</i>	26:1–26:18
Simpler and Better Algorithms for Minimum-Norm Load Balancing <i>Deeparnab Chakrabarty and Chaitanya Swamy</i>	27:1–27:12
On Computing Centroids According to the p -Norms of Hamming Distance Vectors <i>Jiehua Chen, Danny Hermelin, and Manuel Sorge</i>	28:1–28:16
Non-Cooperative Rational Interactive Proofs <i>Jing Chen, Samuel McCauley, and Shikha Singh</i>	29:1–29:16

Stronger ILPs for the Graph Genus Problem <i>Markus Chimani and Tilo Wiedera</i>	30:1–30:15
Complexity of C_k -Coloring in Hereditary Classes of Graphs <i>Maria Chudnovsky, Shenwei Huang, Paweł Rzażewski, Sophie Spirkl, and Mingxian Zhong</i>	31:1–31:15
Consistent Digital Curved Rays and Pseudoline Arrangements <i>Jinhee Chun, Kenya Kikuchi, and Takeshi Tokuyama</i>	32:1–32:16
Efficient Approximation Schemes for Uniform-Cost Clustering Problems in Planar Graphs <i>Vincent Cohen-Addad, Marcin Pilipczuk, and Michał Pilipczuk</i>	33:1–33:14
Improved Bounds for the Excluded-Minor Approximation of Treedepth <i>Wojciech Czerwiński, Wojciech Nadara, and Marcin Pilipczuk</i>	34:1–34:13
Building a Nest by an Automaton <i>Jurek Czyzowicz, Dariusz Dereniowski, and Andrzej Pelc</i>	35:1–35:14
Robustness of Randomized Rumour Spreading <i>Rami Daknama, Konstantinos Panagiotou, and Simon Reisser</i>	36:1–36:15
Structural Rounding: Approximation Algorithms for Graphs Near an Algorithmically Tractable Class <i>Erik D. Demaine, Timothy D. Goodrich, Kyle Kloster, Brian Lavallee, Quanquan C. Liu, Blair D. Sullivan, Ali Vakilian, and Andrew van der Poel</i>	37:1–37:15
Dense Peelable Random Uniform Hypergraphs <i>Martin Dietzfelbinger and Stefan Walzer</i>	38:1–38:16
Efficient Gauss Elimination for Near-Quadratic Matrices with One Short Random Block per Row, with Applications <i>Martin Dietzfelbinger and Stefan Walzer</i>	39:1–39:18
Greedy Strategy Works for k -Center Clustering with Outliers and Coreset Construction <i>Hu Ding, Haikuo Yu, and Zixiu Wang</i>	40:1–40:16
Bidirectional Text Compression in External Memory <i>Patrick Dinklage, Jonas Ellert, Johannes Fischer, Dominik Köppl, and Manuel Penschuck</i>	41:1–41:16
Bisection of Bounded Treewidth Graphs by Convolutions <i>Eduard Eiben, Daniel Lokshantov, and Amer E. Mouawad</i>	42:1–42:11
Oracle-Based Primal-Dual Algorithms for Packing and Covering Semidefinite Programs <i>Khaled Elbassioni and Kazuhisa Makino</i>	43:1–43:15
Online Disjoint Set Cover Without Prior Knowledge <i>Yuval Emek, Adam Goldbraikh, and Erez Kantor</i>	44:1–44:16
Bayesian Generalized Network Design <i>Yuval Emek, Shay Kutten, Ron Lavi, and Yangguang Shi</i>	45:1–45:16

Obviously Strategyproof Mechanisms for Machine Scheduling <i>Diodato Ferraioli, Adrian Meier, Paolo Penna, and Carmine Ventre</i>	46:1–46:15
Going Far From Degeneracy <i>Fedor V. Fomin, Petr A. Golovach, Daniel Lokshantov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi</i>	47:1–47:14
Group Activity Selection with Few Agent Types <i>Robert Ganian, Sebastian Ordyniak, and C. S. Rahul</i>	48:1–48:16
Optimal Sorting with Persistent Comparison Errors <i>Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna</i>	49:1–49:14
Dynamic Dominators and Low-High Orders in DAGs <i>Loukas Georgiadis, Konstantinos Giannis, Giuseppe F. Italiano, Aikaterini Karanasiou, and Luigi Laura</i>	50:1–50:18
On the Hardness and Inapproximability of Recognizing Wheeler Graphs <i>Daniel Gibney and Sharma V. Thankachan</i>	51:1–51:16
Evaluation of a Flow-Based Hypergraph Bipartitioning Algorithm <i>Lars Gottesbüren, Michael Hamann, and Dorothea Wagner</i>	52:1–52:17
Parameterized Approximation Schemes for Independent Set of Rectangles and Geometric Knapsack <i>Fabrizio Grandoni, Stefan Kratsch, and Andreas Wiese</i>	53:1–53:16
Packing Cars into Narrow Roads: PTASs for Limited Supply Highway <i>Fabrizio Grandoni and Andreas Wiese</i>	54:1–54:14
Engineering Negative Cycle Canceling for Wind Farm Cabling <i>Sascha Gritzbach, Torsten Ueckerdt, Dorothea Wagner, Franziska Wegner, and Matthias Wolf</i>	55:1–55:16
Towards Improving Christofides Algorithm for Half-Integer TSP <i>Arash Haddadan and Alantha Newman</i>	56:1–56:12
Counting to Ten with Two Fingers: Compressed Counting with Spiking Neurons <i>Yael Hitron and Merav Parter</i>	57:1–57:17
Triconnected Planar Graphs of Maximum Degree Five are Subhamiltonian <i>Michael Hoffmann and Boris Klemz</i>	58:1–58:14
Parallel Weighted Random Sampling <i>Lorenz Hübschle-Schneider and Peter Sanders</i>	59:1–59:24
External Memory Priority Queues with Decrease-Key and Applications to Graph Algorithms <i>John Iacono, Riko Jacob, and Konstantinos Tsakalidis</i>	60:1–60:14
Shortest Reconfiguration of Perfect Matchings via Alternating Cycles <i>Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto</i>	61:1–61:15
Closing the Gap for Pseudo-Polynomial Strip Packing <i>Klaus Jansen and Malin Rau</i>	62:1–62:14

Odd-Cycle Separation for Maximum Cut and Binary Quadratic Optimization <i>Michael Jünger and Sven Mallach</i>	63:1–63:13
Triangles and Girth in Disk Graphs and Transmission Graphs <i>Haim Kaplan, Katharina Klost, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir</i>	64:1–64:14
Reliable Hubs for Partially-Dynamic All-Pairs Shortest Paths in Directed Graphs <i>Adam Karczmarz and Jakub Łącki</i>	65:1–65:15
Min-Cost Flow in Unit-Capacity Planar Graphs <i>Adam Karczmarz and Piotr Sankowski</i>	66:1–66:17
Global Curve Simplification <i>Mees van de Kerkhof, Irina Kostitsyna, Maarten Löffler, Majid Mirzanezhad, and Carola Wenk</i>	67:1–67:14
Trace Reconstruction: Generalized and Parameterized <i>Akshay Krishnamurthy, Arya Mazumdar, Andrew McGregor, and Soumyabrata Pal</i>	68:1–68:25
Generalized Assignment via Submodular Optimization with Reserved Capacity <i>Ariel Kulik, Kanthi Sarpatwar, Baruch Schieber, and Hadas Shachnai</i>	69:1–69:15
Resilient Dictionaries for Randomly Unreliable Memory <i>Stefano Leucci, Chih-Hung Liu, and Simon Meierhans</i>	70:1–70:16
Hermitian Laplacians and a Cheeger Inequality for the Max-2-Lin Problem <i>Huan Li, He Sun, and Luca Zanetti</i>	71:1–71:14
Packing Directed Circuits Quarter-Integrally <i>Tomáš Masařík, Irene Muzi, Marcin Pilipczuk, Paweł Rzażewski, and Manuel Sorge</i>	72:1–72:13
Equal-Subset-Sum Faster Than the Meet-in-the-Middle <i>Marcin Mucha, Jesper Nederlof, Jakub Pawlewicz, and Karol Węgrzycki</i>	73:1–73:16
Hardness of Bichromatic Closest Pair with Jaccard Similarity <i>Rasmus Pagh, Nina Mesing Stausholm, and Mikkel Thorup</i>	74:1–74:13
Compact Oblivious Routing <i>Harald Räcke and Stefan Schmid</i>	75:1–75:14
Geometric Crossing-Minimization – A Scalable Randomized Approach <i>Marcel Radermacher and Ignaz Rutter</i>	76:1–76:16
An Approximate Kernel for Connected Feedback Vertex Set <i>M. S. Ramanujan</i>	77:1–77:14
Multicommodity Multicast, Wireless and Fast <i>R. Ravi and Oleksandr Rudenko</i>	78:1–78:20
Recognizing Planar Laman Graphs <i>Jonathan Rollin, Lena Schlipf, and André Schulz</i>	79:1–79:12
Simultaneous Representation of Proper and Unit Interval Graphs <i>Ignaz Rutter, Darren Strash, Peter Stumpf, and Michael Vollmer</i>	80:1–80:15
Correlation Clustering with Same-Cluster Queries Bounded by Optimal Cost <i>Barna Saha and Sanjay Subramanian</i>	81:1–81:17

0:x **Contents**

Graph Balancing with Orientation Costs <i>Roy Schwartz and Ran Yeheskel</i>	82:1–82:15
FPT-Algorithms for Computing Gromov-Hausdorff and Interleaving Distances Between Trees <i>Elena Farahbakhsh Touli and Yusu Wang</i>	83:1–83:14

■ Preface

This volume contains the extended abstracts selected for presentation at ESA 2019, the 27th European Symposium on Algorithms, held in Munich/Garching, Germany, on 9–11 September 2019, as part of ALGO 2019. 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 2019, 329 papers were submitted, 276 for Track A and 53 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 83 papers for inclusion in the program, 70 from track A and 13 from track B, yielding an acceptance rate of about 25%.

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 Cornelius Brand for the paper “Patching Colors with Tensors”. The best paper award for track A was given to Peyman Afshani, Rolf Fagerberg, David Hammer, Riko Jacob, Irina Kostitsyna, Ulrich Meyer, Manuel Penschuck and Nodari Sitchinava for the paper “Fragile Complexity of Comparison-Based Algorithms”. The best paper award for track B was given to Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, Ulrich Meyer, Manuel Penschuck and Christopher Weyand for the paper “Efficiently Generating Geometric Inhomogeneous and Hyperbolic Random Graphs”. No best student paper award has been given this year.

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

- Aditya Bhaskara, University of Utah
- Sayan Bhattacharya, University of Warwick
- Sebastian Brandt, ETHZ
- Raphaël Clifford, University of Bristol
- Éric Colin de Verdière, CNRS and Université Paris-Est
- Moran Feldman, The Open University of Israel
- Jugal Garg, University of Illinois at Urbana-Champaign
- Sariel Har-Peled, University of Illinois at Urbana-Champaign
- Pinar Heggernes, University of Bergen
- Stacey Jeffery, CWI
- Sagar Kale, EPFL
- Matya Katz, Ben-Gurion University
- Tsvi Kopelowitz, Bar-Ilan University
- Bundit Laekhanukit, Shanghai University of Finance and Economics
- Silvio Lattanzi, Google
- Pasin Manurangsi, UC Berkeley
- Claire Mathieu, CNRS, Paris
- Nicole Megow, University of Bremen
- Shay Moran, Princeton University
- Rad Niazadeh, Stanford University
- Ely Porat, Bar-Ilan University
- Eva Rotenberg, Technical University of Denmark
- Aviad Rubinfeld, Stanford University
- Barna Saha, University of Massachusetts Amherst
- Pascal Schweitzer, TU Kaiserslautern
- David Shmoys, Cornell University
- Sahil Singla, Princeton University
- Cliff Stein, Columbia University
- Ola Svensson, EPFL, (PC Chair)
- Aravindan Vijayaraghavan, Northwestern University
- Magnus Wahlström, Royal Holloway, University of London
- Justin Ward, Queen Mary University of London
- Standa Živný, University of Oxford



Track B (Engineering and Applications) Program Committee

- Dan Alistarh, Institute of Science and Technology Austria (IST Austria)
- Michael A. Bender, Stony Brook University, (PC Chair)
- Anne Benoit, Ecole Normale Supérieure de Lyon
- Jon Berry, Sandia National Laboratories
- Timo Bingmann, Karlsruhe Institute of Technology
- Kevin Buchin, TU Eindhoven
- Matteo Ceccarello, IT University of Copenhagen and BARC
- Martín Farach-Colton, Rutgers University
- Rati Gelashvili, NeuralMagic
- Seth Gilbert, National University of Singapore
- Piyush Kumar, Florida State University
- Rob Johnson, VMware Research
- Irina Kostitsyna, TU Eindhoven
- Jing Li, New Jersey Institute of Technology
- Samuel McCauley, Bar Ilan University
- John Owens, UC Davis
- Prashant Pandey, Carnegie Mellon University
- Tao B. Schardl, MIT
- Julian Shun, MIT
- Francesco Silvestri, University of Padova
- Sebastian Wild, University of Waterloo
- Maxwell Young, Mississippi State University

■ List of External Reviewers

Mohammad Ali Abam
Sepehr Abbasi Zadeh
Karim Abu-Affash
Huseyin Acan
Raghavendra Addanki
David Adjiashvili
Peyman Afshani
Akanksha Agrawal
Oswin Aichholzer
Yaroslav Akhremtsev
Vitaly Aksenov
Jarno Alanko
Yeganeh Alimohammadi
Josh Alman
Matteo Almanza
Andreas Alpers
Amihoud Amir
Hyung-Chan An
Haris Angelidakis
Patrizio Angelini
Spyros Angelopoulos
Antonios Antoniadis
Charles J. Argue
Eyjólfur Ingi Ásgeirsson
Saman Ashkiani
James Aspnes
Sepehr Assadi
Martin Aumüller
Brian Axelrod
Yossi Azar
Arturs Backurs
Maryam Bahrani
Alkida Balliu
Aritra Banik
Max Bannach
Nikhil Bansal
Amitabh Basu
Michael Bekos
Rémy Belmonte
Aleksandrs Belovs
Robin Belton
Naama Ben-David
Huck Bennett
Sergey Bereg
Sebastian Berndt
Hedyeh Beyhaghi
Amey Bhangale
Umang Bhaskar
Arnab Bhattacharyya
Marcin Bienkowski
Philip Bille
Vittorio Bilò
Greg Bodwin
Martin Böhm
Nicolas Bonichon
Flavia Bonomo
Prosenjit Bose
Marin Bougeret
Nicolas Bousquet
Joshua Brakensiek
Robert Brederick
Marco Bressan
Karl Bringmann
Benjamin Brock
Trevor Brown
Guido Brückner
Niv Buchbinder
Mickaël Buchet
Maïke Buchin
Lilian Buzer
Jarek Byrka
Sergio Cabello
Yixin Cao
Jean Cardinal
John Carlsson
Deeparnab Chakrabarty
Diptarka Chakraborty
Parinya Chalermsook
Erin Chambers
Timothy M. Chan
Hsien-Chih Chang
Panagiotis Charalampopoulos
Vaggos Chatziafratis
Bernard Chazelle
Jiehua Chen
Lijie Chen
Lin Chen
Louis Chen
Xue Chen
Yang Chen

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

0:xvi List of External Reviewers

Yijia Chen
Flavio Chierichetti
Ashish Chiplunkar
Rajesh Chitnis
Man-Kwun Chiu
Eden Chlamtac
Chi-Ning Chou
Keerti Choudhary
David Coeurjolly
Vincent Cohen-Addad
Alessio Conte
Alex Conway
Martin Cooper
Jose Correa
Christophe Crespelle
Ágnes Cseh
Monika Csikos
Marek Cygan
Giordano Da Lozzo
Yuval Dagan
Mina Dalirrooyfard
Debarati Das
Sandip Das
Syamantak Das
Anindya De
Paloma de Lima
Arnaud de Mesmay
Minati De
Mateus De Oliveira Oliveira
Michel de Rougemont
Dan DeBlasio
Oscar Defrain
Holger Dell
Mahsa Derakhshan
Olivier Devillers
Laxman Dhulipala
Martin Dietzfelbinger
Michael Dinitz
Yann Disser
Shahar Dobzinski
Benjamin Doerr
Simon Doherty
Mitre Dourado
Anne Driemel
Paul Duetting
Vedran Dunjko
Christoph Dürr
Stefan Edelkamp
Alon Efrat
Eduard Eiben
Friedrich Eisenbrand
Marek Elias
Alessandro Epasto
Leah Epstein
Jeff Erickson
Thomas Erlebach
Bruno Escoffier
Hossein Esfandiari
Meryem Essaidi
Ruy Fabila-Monroy
Yuri Faenza
Rolf Fagerberg
Matthew Fahrback
Chenglin Fan
Angelo Fanelli
Lene Favrholdt
Carl Feghali
Michal Feldman
Yiding Feng
Hendrik Fichtenberger
Omrit Filtser
Jeremy Fineman
Manuela Fischer
Fedor Fomin
Sebastian Forster
Kyle Fox
Zachary Friggstad
Daniel Frishberg
Radoslav Fulek
Martin Fürer
Travis Gagie
Jakub Gajarský
Sainyam Galhotra
Buddhima Gamlath
Robert Ganian
Mohit Garg
Naveen Garg
Nikhil Garg
Luisa Gargano
Cyril Gavoille
Pawel Gawrychowski
Afton Geil
Suprovat Ghoshal
Vasilis Gkatzelis
Marc Glisse
Xavier Goaoc
Shay Golan
Mordecai J. Golin

Petr Golovach	Dominik Kaaser
Mayank Goswami	Tim Kaler
Lee-Ad Gottlieb	Frank Kammer
Allan Grønlund	Mamadou Moustapha Kanté
Yan Gu	Karthik C. S.
Joachim Gudmundsson	Telikepalli Kavitha
Zhishan Guo	Kamer Kaya
Manoj Gupta	Mark Keil
Sushmita Gupta	Steven Kelk
Varun Gupta	Nathaniel Kell
Gregory Gutin	Dominik Kempa
Waldo Gálvez	Yukiko Kenmochi
Shahrazad Haddadan	Kristen Kessel
Torben Hagerup	Thomas Kesselheim
Mohammadtaghi Hajiaghayi	Arindam Khan
Dan Halperin	Seri Khoury
Oussama Hanguir	Samir Khuller
Tero Harju	Philipp Kindermann
Tobias Harks	Evangelos Kipouridis
Aram Harrow	Fredrik Berg Kjolstad
Meike Hatzel	Dušan Knop
Qizheng He	Sang-Ki Ko
Peter Hegarty	Tomasz Kociumaka
Monika Henzinger	Jochen Koenemann
D. Ellis Hershkowitz	Jukka Kohonen
Ruben Hoeksma	Sudeshna Kolay
Markus Holzer	Christian Komusiewicz
Chien-Chung Huang	Christian Konrad
Pavel Hubáček	Aryeh Kontorovich
Thore Husfeldt	Janne H. Korhonen
Edin Husic	Guy Kortsarz
Tomohiro I	Arie Koster
David Ilcinkas	Grammateia Kotsialou
Kazuo Iwama	Lukasz Kowalik
Taisuke Izumi	Laszlo Kozma
Riko Jacob	Jan Kratochvil
Lars Jaffke	Stefan Kratsch
Arun Jambulapati	Philipp Klaus Krause
Ravi Janardan	Ravishankar Krishnaswamy
Bart M. P. Jansen	Fabian Kuhn
Klaus Jansen	Alan Kuhnle
Jesper Jansson	Janardhan Kulkarni
Haotian Jiang	Pooja Kulkarni
Shaofeng Jiang	Rucha Kulkarni
Ce Jin	Amit Kumar
Gorav Jindal	Florian Kurpicz
Matthew Johnson	William Kuzmaul
Vincent Jugé	Martin Kutrib

0:xviii List of External Reviewers

Tsz Chiu Kwok	Peter McGlaughlin
O-Joung Kwon	Kitty Meeks
Thijs Laarhoven	Julian Mestre
Chi-Kit Lam	Othon Michail
Sebastian Lamm	Theresa Migler-Vondollen
Michael Lampis	David Miller
Barbara Langfeld	Till Miltzow
John Lapinskas	Majid Mirzanezhad
Thomas Lavastida	Neeldhara Misra
Francis Lazarus	Pranabendu Misra
Euiwoong Lee	Slobodan Mitrovic
James Lee	Shuichi Miyazaki
Yin Tat Lee	Matthias Mnich
Stefano Leucci	Divyarthi Mohan
Roie Levin	Tobias Mömke
Avivit Levy	Ashley Montanaro
Caleb Levy	Pedro Montealegre
Jason Li	Benjamin Moore
Weidong Li	Shlomo Moran
Yingkai Li	Benjamin Moseley
Chao Liao	Guy Moshkovitz
Nutan Limaye	Amer Mouawad
Bingkai Lin	Moritz Mühlenthaler
Andre Linhares	Tathagata Mukherjee
Paul Liu	Wolfgang Mulzer
Maarten Löffler	Ian Munro
Kefu Lu	Ahad N. Zehmakan
Anna Lubiw	Wojciech Nadara
Ben Lund	Viswanath Nagarajan
Agnieszka Łupińska	Kotaro Nakagawa
Vivek Madan	Yuto Nakashima
Sepideh Mahabadi	Subhas Nandy
Diptapriyo Majumdar	Seffi Naor
Konstantin Makarychev	N.S. Narayanaswamy
Yury Makarychev	Emanuele Natale
Veli Mäkinen	Abhinandan Nath
Michalis Mamakos	Amir Nayyeri
Fredrik Manne	Jesper Nederlof
Jieming Mao	Yakov Nekrich
Guillaume Marçais	Daniel Neuen
Mathieu Mari	André Nichterlein
Clément Maria	Prajakta Nimbhorkar
Barnaby Martin	Martin Nöllenburg
Dániel Marx	Ashkan Norouzi Fard
Jannik Matuschke	Zeev Nutov
Alexander Matveev	Eunjin Oh
Simon Mauras	Yoshio Okamoto
Yannic Maus	Dániel Oláh

Dennis Olivetti
Neil Olver
Krzysztof Onak
Tim Ophelders
Sebastian Ordyniak
Sang-Il Oum
Eli Packer
Rasmus Pagh
Konstantinos Panagiotou
Fahad Panolan
Charis Papadopoulos
Biswas Parajuli
Nikos Parotsidis
Manuel Penschuck
David Phillips
Jeff Phillips
Stephen Piddock
Marcin Pilipczuk
Solon Pissis
Benjamin Plaut
Kirk Pruhs
Simon Puglisi
Manish Purohit
Ruixin Qiang
Kent Quanrud
Jaikumar Radhakrishnan
Sharath Raghvendra
Saladi Rahul
Benjamin Raichel
Shravas Rao
Cyrus Rashtchian
R Ravi
Bhaskar Ray Chaudhury
Jean-Florent Raymond
Ilya Razenshteyn
Damien Regnault
Daniel Reichman
Marc Renault
Alireza Rezaei
Havana Rika
Liam Roditty
Marcel Roeloffzen
Lars Rohwedder
J r mie Roland
Jonathan Rollin
Miguel Romero Orth
Benjamin Rossman
Peter Rossmanith
Marc Roth
Thomas Rothvoss
Ahana Roy Choudhury
Mikhail Rudoy
Ignaz Rutter
Joel Rybicki
Sushant Sachdeva
Andrej Sajenko
Gelasio Salazar
Domenico Salvagnin
Peter Sanders
Laura Sanit 
Thatchaphol Saranurak
Kanthi Sarpatwar
Saket Saurabh
Raghuvansh Saxena
Dominik Scheder
Kevin Schewior
Andreas Schmid
Dominik Schreiber
Marc Schr der
Andr  Schulz
Ariel Schwartzman
Roy Schwartz
Gregory Schwartzman
Chris Schwiegelshohn
Saeed Seddighin
C. Seshadhri
Martin P. Seybold
Jiř  Sgall
Alkmini Sgouritsa
Rahul Shah
Amirbehshad Shahrabi
Riva Shalom
Ali Shameli
Dana Shapira
Or Sheffet
Xiangkun Shen
Jessica Shi
Anastasios Sidiropoulos
Florian Sikora
Harsha Vardhan Simhadri
Paris Siminelakis
Bertrand Simon
Kirill Simonov
Mohit Singh
Shikha Singh
Rene Sitters

Nolan Skochdopole
Piotr Skowron
Jack Snoeyink
Shay Solomon
Ramanujan M. Sridharan
Frank Staals
Tatiana Starikovskaya
Sabine Storaandt
Leen Stougie
Darren Strash
Torstein Strømme
Anand Subramanian
Sanjay Subramanian
Ondrej Suchy
Pattara Sukprasert
Xiaorui Sun
Yihan Sun
Cynthia Sung
Akira Suzuki
Zoya Svitkina
Chaitanya Swamy
Wai Ming Tai
Maurizio Talamo
Prafullkumar Tale
Ohad Talmon
Martin Tancer
Zhihao Gavin Tang
Biaoshuai Tao
Anusch Taraz
Jan Arne Telle
Sharma V. Thankachan
Louis Theran
Sumedh Tirodkar
Dave Touchette
Dekel Tsur
Jara Uitto
William Umboh
Sumedha Uniyal
Robert Utterback
Przemysław Uznański
Rene van Bevern
Jan van den Heuvel
Arthur van Goethem
André van Renssen
Rob van Stee
Anke van Zuylen
Kasturi Varadarajan
Sergei Vassilvitskii
Daniel Vaz
Rahul Vaze
Suresh Venkatasubramanian
José Verschae
Adrian Vladu
Paola Vocca
Ben Lee Volk
Moritz von Looz
Bernhard von Stengel
Alexandros Voudouris
Hoa Vu
Nikhil Vyas
Tal Wagner
Erik Waingarten
David Wajc
Haitao Wang
Joshua Wang
Lusheng Wang
Yipu Wang
Yusu Wang
Karol Węgrzycki
Zhewei Wei
Oren Weimann
Nicole Wein
S. Matthew Weinberg
Armin Weiß
Colin White
Andreas Wiese
Alexander Wolff
Prudence Wong
John Wright
Marcin Wrochna
Christian Wulff-Nilsen
Allen Xiao
Helen Xu
Pan Xu
Mobin Yahyazadeh
Hadi Yami
Lin Yang
Yuichi Yoshida
Viktor Zamaraev
Or Zamir
Alireza Zarei
Meirav Zehavi
Yuhao Zhang
Mingxian Zhong
Hang Zhou
Samson Zhou
Aleksandar Zlateski

Constant-Factor FPT Approximation for Capacitated k -Median

Marek Adamczyk

University of Warsaw, Poland
marek.adamczyk@mimuw.edu.pl

Jarosław Byrka

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

Jan Marcinkowski 

University of Wrocław, Poland
jan.marcinkowski@cs.uni.wroc.pl

Syed M. Meesum

University of Wrocław, Poland
syedmohammad.meesum@uwr.edu.pl

Michał Włodarczyk 

University of Warsaw, Poland
m.wlodarczyk@mimuw.edu.pl

Abstract

Capacitated k -median is one of the few outstanding optimization problems for which the existence of a polynomial time constant factor approximation algorithm remains an open problem. In a series of recent papers algorithms producing solutions violating either the number of facilities or the capacity by a multiplicative factor were obtained. However, to produce solutions without violations appears to be hard and potentially requires different algorithmic techniques. Notably, if parameterized by the number of facilities k , the problem is also $W[2]$ hard, making the existence of an exact FPT algorithm unlikely. In this work we provide an FPT-time constant factor approximation algorithm preserving both cardinality and capacity of the facilities. The algorithm runs in time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$ and achieves an approximation ratio of $7 + \varepsilon$.

2012 ACM Subject Classification Theory of computation \rightarrow Facility location and clustering; Theory of computation \rightarrow Fixed parameter tractability

Keywords and phrases K -median, Clustering, Approximation Algorithms, Fixed Parameter Tractability

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.1

Funding *Jarosław Byrka*: Supported by the Polish National Science Centre grant 2015/18/E/ST6/00456.

Jan Marcinkowski: Supported by the Polish National Science Centre grant 2015/18/E/ST6/00456.

Syed M. Meesum: Supported by the Polish National Science Centre grant 2015/18/E/ST6/00456.

1 Introduction

For many years approximation algorithms and FPT algorithms were developed in parallel. Recently the two paradigms are being combined and provide intriguing discoveries in the intersection of the two worlds. It is particularly interesting in the case of problems for which we fail to make progress on improving the approximation ratios in polynomial time. An excellent example of such a combination is the FPT approximation algorithm for the k -CUT problem by Gupta et al. [17].



© Marek Adamczyk, Jarosław Byrka, Jan Marcinkowski, Syed M. Meesum, and Michał Włodarczyk; licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 1; pp. 1:1–1:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this work we focus on the CAPACITATED k -MEDIAN problem, whose approximability attracted attention of many researchers. Unlike in the case of the k -CUT problem, it is still not clear what approximation is possible for CAPACITATED k -MEDIAN in polynomial time. As shall be discussed in more detail in the following section, the best true approximation known is $\mathcal{O}(\log k)$ based on tree embedding of the underlying metric. The other algorithms either violate the bound on the number of facilities or the capacity constraints.

Our main result is a $(7 + \epsilon)$ -approximation algorithm for the CAPACITATED k -MEDIAN problem running in $\text{FPT}(k)$ time, that exploits techniques from both – approximation and FPT – realms. The algorithm builds on the idea of clustering the clients into $\ell = \mathcal{O}(k \cdot (\log n)/\epsilon)$ locations, which is similar to the approach from the $\mathcal{O}(\log k)$ -approximation algorithm, where one creates $\mathcal{O}(k)$ clusters. This is followed by guessing the distribution of the k facilities inside these ℓ clusters. Having such a structure revealed, we simplify the instance further by rounding particular distances and reduce the problem to linear programming over a totally unimodular matrix.

1.1 Problems overview and previous work

In the CAPACITATED k -MEDIAN problem (CKM), we are given a set F of facilities, each facility f with a capacity $u_f \in \mathbb{Z}_{\geq 0}$, a set C of clients, a metric d over $F \cup C$ and an upper bound k on the number of facilities we can open. A solution to the CKM problem is a set $S \subseteq F$ of at most k open facilities and a connection assignment $\phi : C \rightarrow S$ of clients to open facilities such that $|\phi^{-1}(f)| \leq u_f$ for every facility $f \in S$. The goal of the problem is to find a solution that minimizes the connection cost $\sum_{c \in C} d(c, \phi(c))$. In the case when all the facilities can serve at most u clients, for some integer u , we obtain the UNIFORM CKM problem.

Uncapacitated k -median. The standard k -median problem, where there is no restriction on the number of clients served by a facility, can be approximated up to a constant factor [9, 2]. The current best is the $(2.675 + \epsilon)$ -approximation algorithm of Byrka et al. [4], which is a result of optimizing a part of the algorithm by Li and Svensson [23].

Approximability of CKM. As already stressed, CAPACITATED k -MEDIAN is among few remaining fundamental optimization problems for which it is not clear if there exist polynomial time constant factor approximation algorithms. All the known algorithms violate either the number of facilities or the capacities. In particular, already the algorithm of Charikar et al. [9] gave 16-approximate solution for the uniform capacitated k -median violating the capacities by a factor of 3. Then Chuzhoy and Rabani [10] considered general capacities and gave a 50-approximation algorithm violating capacities by a factor of 40.

The difficulty appears to be related to the unbounded integrality gap of the standard LP relaxation. To obtain integral solutions that are bounded with respect to the fractional solution to the standard LP, one has to either allow the integral solution to open twice as many facilities or to violate the capacities by a factor of two. LP-rounding algorithms essentially matching these limits have been obtained [1, 3].

Subsequently, Li broke this integrality gap barrier by giving a constant factor algorithm for the capacitated k -median by opening $(1 + \epsilon) \cdot k$ facilities [21, 22]. Afterwards analogous results, but violating the capacities by a factor of $(1 + \epsilon)$ were also obtained [5, 14].

The algorithms with $(1 + \epsilon)$ violations are all based on strong LP relaxations containing additional constraints for subsets of facilities. Notably, it is not clear if these relaxations can be solved exactly in polynomial time, still they suffice to construct an approximation algorithm via the “round-or-separate” technique that iteratively adds consistency constraints

for selected subsets. Although while spectacularly breaking the standard LP integrality bound, these techniques appear insufficient to yield a proper approximation algorithm that does not violate constraints.

The only true approximation for CKM known is a folklore $\mathcal{O}(\log k)$ approximation algorithm that can be obtained via the metric tree embedding with expected logarithmic distortion [15]. To the best of our knowledge, this result has not been explicitly published, but it can be obtained similarly to the $\mathcal{O}(\log k)$ -approximation for UNCAPACITATED KM by Charikar [7]. For the sake of completeness and since it follows easily from our framework, we give its proof in Section 3 without claiming credit for it. This $\mathcal{O}(\log k)$ barrier is in contrast with other capacitated clustering problems such as facility location and k -center, for which constant factor approximation algorithms are known [19, 12].

After our work was announced, Xu et al. [27] proposed a similar algorithm for Euclidean CAPACITATED k -MEANS, i.e., a constant factor approximation running in time $\text{FPT}(k)$. Both our and their approximation ratios have been very recently improved by Cohen-Addad and Li [11], who obtained $(3 + \varepsilon)$ for CAPACITATED k -MEDIAN and $(9 + \varepsilon)$ for CAPACITATED k -MEANS in general metric spaces. They have also provided a deeper insight into the problems basing on the framework of coresets.

1.2 Parameterized Complexity

A parameterized problem instance is created by associating an input instance with an integer parameter k . We say that a problem is *fixed parameter tractable* (FPT) if every instance (I, k) of the problem can be solved in time $f(k) \cdot |I|^{\mathcal{O}(1)}$, where f is an arbitrary computable function of k .

We say that a problem is FPT if it is possible to give an algorithm that solves it in running time of the required form. Such an algorithm we shall call a *parameterized algorithm*.

To show that a problem is unlikely to be FPT, we use parameterized reductions analogous to those employed in the classical complexity theory. Here, the concept of W -hardness replaces the one of NP-hardness, and we need not only construct an equivalent instance in FPT time, but also ensure that the size of the parameter in the new instance depends only on the size of the parameter in the original instance. In contrast to the NP-hardness theory, there is a hierarchy of classes $\text{FPT} = W[0] \subseteq W[1] \subseteq W[2] \subseteq \dots$ and these containments are believed to be strict. If there exists a parameterized reduction transforming a problem known to be $W[t]$ -hard for $t > 0$ to another problem Π , then the problem Π is $W[t]$ -hard as well. This provides an argument that Π is unlikely to admit an algorithm with running time $f(k) \cdot |I|^{\mathcal{O}(1)}$.

We begin with an argument that allowing FPT time for (even uncapacitated) k -MEDIAN should not help in finding the optimal solution and we still need to settle for approximation.

▷ **Fact 1.** The UNCAPACITATED k -MEDIAN problem is $W[2]$ -hard when parameterized by k , even on metrics induced by unweighted graphs.

Proof. Consider an instance (G, k) of the DOMINATING SET problem, which is $W[2]$ -hard when parameterized by the solution size k . Graph G induces a metric such that the distance between two adjacent vertices equals one and otherwise the distance between vertices is the length of the shortest path. A dominating set of size at most k exists in graph G if and only if we can find a vertex set S of size k , such that all the other vertices are at distance 1 from S . This is equivalent to the solution to UNCAPACITATED k -MEDIAN on the metric induced by G being of size exactly $|V(G)| - k$. ◁

Parameterized Approximation

In recent years new research directions emerged in the intersection of the theory of approximation algorithms and the FPT theory. It turned out that for some problems that are intractable in the exact sense, parameterization still comes in useful when we want to reduce the approximation ratio. Some examples are $(2 - \varepsilon)$ -approximation for k -CUT [17] or $f(\mathcal{F})$ -approximation for PLANAR- \mathcal{F} DELETION [16] for some implicit function f . The dependency on \mathcal{F} was later improved, leading to $\mathcal{O}(\log k)$ -approximations for, e.g., k -VERTEX SEPARATOR [20] and k -TREEWIDTH DELETION [18].

On the other hand some problems parameterized by the solution size have been proven resistant to such improvements. Chalermsook et al. [6] observed that under the assumption of Gap-ETH there can be no parameterized approximation with ratio $o(k)$ for k -CLIQUE and none with ratio $f(k)$ for k -DOMINATING SET (for any function f). Subsequently Gap-ETH has been replaced with a better established hardness assumption $\text{FPT} \neq \text{W}[1]$ for k -DOMINATING SET [25].

1.3 Organization of the paper

Our main result is stated in Theorem 16 (Section 4.3), where we present a $(7 + \varepsilon)$ -approximation algorithm for the NON-UNIFORM CKM problem running in $\text{FPT}(k)$ time.

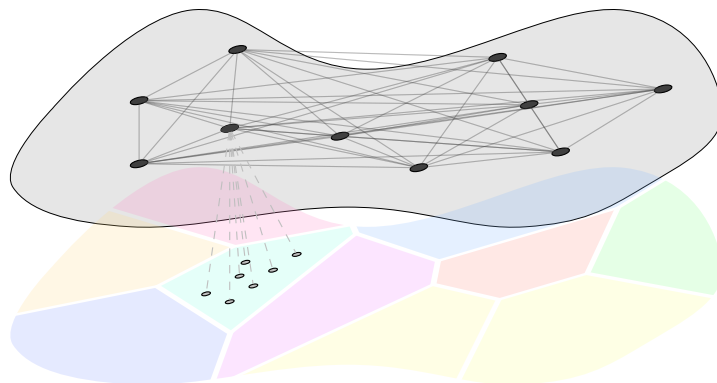
To obtain this result we need two ingredients. The first one is a metric embedding that reduces the problem to a simpler instance, called ℓ -centered, which is described in Section 2. This reduction provides a richer structure, which can be exploited to obtain the folklore $\mathcal{O}(\log k)$ -approximation via tree embeddings [15]. As already mentioned, similar approach was presented by Charikar et al. [7] in their algorithm for the uncapacitated setting. For the sake of completeness, we present this result in Appendix 3.

The second ingredient is a parameterized algorithm for the ℓ -centered instances. Since it is simpler in the uniform setting, we first solve it in Section 4.2 as a warm up before the main result. This way the new ideas are being revealed gradually to the reader.

2 ℓ -Centered instances

Suppose we work with a graph on nodes $F \cup C$, on which we are given a metric d . In our considerations the set $F \cup C$ will be fixed throughout, however we will be modifying the metric over it. Consider an algorithm ALG which produces a solution $ALG(d)$ for a metric d . This solution can be seen as a mapping which we explicitly denote by $\phi^{ALG(d)}$. Its cost in the metric d' equals $\sum_{c \in C} d'(c, \phi^{ALG(d)})$ which we shall briefly denote by $\text{cost}(\phi^{ALG(d)}, d')$. The second argument is useful, when an algorithm ALG produces a solution (mapping) $ALG(d)$ with respect to metric d , but later on we may be interested in its cost over a different metric. Also, let $OPT(d)$ denote the optimum solution for the CKM problem on metric d .

In order to solve CKM, we shall invoke an algorithm for UNCAPACITATED KM as a subroutine. Let $ALG_{unc}^\ell(d)$ be a relaxed solution that opens up to $\ell \geq k$ facilities and can break the capacity constraints. It induces a mapping which, for consistency, we shall denote by $\phi^{ALG_{unc}^\ell(d)}$. Observe that in this mapping every client can be connected to the closest open facility. Since UNCAPACITATED KM admits constant approximation algorithms, we can work with solutions satisfying: $\text{cost}(\phi^{ALG_{unc}^\ell(d)}, d) = O(\text{cost}(\phi^{OPT(d)}, d))$. The larger ℓ we allow in the relaxation, the smaller constant we will be able to achieve in the relation above.



■ **Figure 1** An ℓ -centered instance. In the upper layer there is a set S of ℓ vertices connected as a clique. The rest of vertices are divided into separate clusters. Vertices in a single cluster are only connected to their center in the set S .

Using such an algorithm for UNCAPACITATED KM as a subroutine, we can find a simpler metric to work with. First we build a graph which will induce the metric. Let $F(\text{ALG}_{unc}^\ell(d))$ be the set of facilities opened by $\text{ALG}_{unc}^\ell(d)$. For each such a facility f we create a copy vertex s^f , which is at distance 0 from f . We denote the set of copies by S , i.e., $S = \{s^f \mid f \in F(\text{ALG}_{unc}^\ell(d))\}$. Given that we demand the distance from f to s^f to be 0, we can naturally extend the metric d to the set $C \cup F \cup S$. To distinguish facilities from $F(\text{ALG}_{unc}^\ell(d))$ from their copies S , we shall call each copy $s \in S$ a *center*.

We build a complete graph on S and preserve the metric d therein. For every node $v \notin S$, be it either a client from C or a facility from F , we place an edge to the closest (according to the extended d) center $s^v \in S$ and set its length to $d(v, s^v)$. We call such a graph ℓ -centered and refer to its induced metric as d_ℓ .

► **Definition 2.** An instance of CKM is called ℓ -centered if the metric, which we shall denote by d_ℓ , is induced by a weighted graph $G(F \cup C \cup S, E)$ such that

1. $|S| \leq \ell$,
2. $\binom{S}{2} \subseteq E$, i.e., S forms a clique,
3. for every $v \in C \cup F$ there is only one edge incident to v in E , and it connects v to some $s^v \in S$.

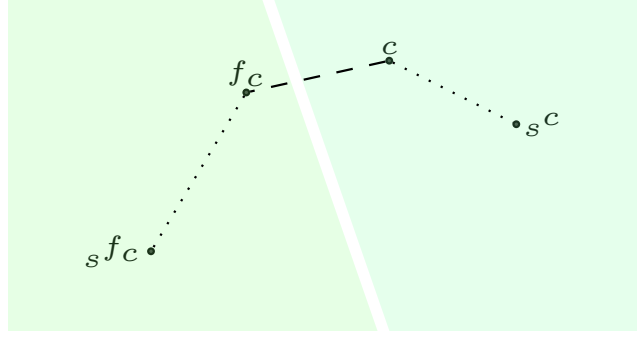
For a center $s \in S$ we shall say that all nodes from $F \cup C$ that are connected to s form a *cluster* of s . If we consider only nodes from F , then we talk about an f -cluster of s , denoted $F(s)$.

The idea of preprocessing that simplifies the metric by recognizing a small number of hubs resembles the notion of α -preserving metrics, that have been used as a tool to obtain coresets for the related problem BALANCED k -MEDIAN [13].

In the following lemma we relate the cost of embedding the optimum solution $\text{OPT}(d)$ from a metric d to d_ℓ .

► **Lemma 3** (Embedding d into ℓ -centered metric d_ℓ). Let $\text{ALG}_{unc}^\ell(d)$ be a solution for the UNCAPACITATED KM problem on metric d from which we construct the ℓ -centered instance. Optimal solution $\text{OPT}(d)$ can be embedded into an ℓ -centered metric d_ℓ with the cost relation being

$$\text{cost}(\phi^{\text{OPT}(d)}, d) \leq \text{cost}(\phi^{\text{OPT}(d)}, d_\ell) \leq 3 \cdot \text{cost}(\phi^{\text{OPT}(d)}, d) + 4 \cdot \text{cost}(\phi^{\text{ALG}_{unc}^\ell(d)}, d).$$



■ **Figure 2** Situation in Lemma 3. In the optimum solution to the CKM instance, client c is connected to the facility f_c . In the ℓ -centered instance c resides in a cell, where s^c is a center. The center of f_c is s^{f_c} .

Proof. Let c be a client connected to facility f_c in the optimal solution $OPT(d)$. Let s^c be the center closest to c within S (the ℓ -center), and let s^{f_c} be the center closest to f_c . First let us note that $d_\ell(c, f_c) = d(c, s^c) + d(s^c, s^{f_c}) + d(s^{f_c}, f_c)$. Next we bound the terms $d(f_c, s^{f_c})$ and $d(s^c, s^{f_c})$ separately.

▷ **Fact 4.** For every client c and its facility f_c from OPT we have $d(f_c, s^{f_c}) \leq d(f_c, c) + d(c, s^c)$.

Proof. Since s^{f_c} is the closest ℓ -center to the facility f_c , we have that $d(f_c, s^{f_c}) \leq d(f_c, s^c)$. At the same time, from the triangle inequality it follows that $d(f_c, s^c) \leq d(f_c, c) + d(c, s^c)$. ◁

▷ **Fact 5.** For each c we have $d(s^c, s^{f_c}) \leq 2(d(f_c, c) + d(c, s^c))$.

Proof. From the triangle inequality we know that

$$d(s^c, s^{f_c}) \leq d(s^c, c) + d(c, f_c) + d(f_c, s^{f_c}).$$

From Fact 4 we also know that $d(f_c, s^{f_c}) \leq d(f_c, c) + d(c, s^c)$, and combining the two inequalities we get $d(s^c, s^{f_c}) \leq d(s^c, c) + d(c, f_c) + d(f_c, s^{f_c}) \leq 2(d(f_c, c) + d(c, s^c))$. ◁

These two facts imply

$$\begin{aligned} d_\ell(c, f_c) &= d(c, s^c) + d(s^c, s^{f_c}) + d(s^{f_c}, f_c) \\ &\leq d(c, s^c) + d(s^c, s^{f_c}) + (d(f_c, c) + d(c, s^c)) && \text{(from Fact 4)} \\ &\leq d(c, s^c) + 2(d(f_c, c) + d(c, s^c)) + (d(f_c, c) + d(c, s^c)) && \text{(from Fact 5)} \\ &= 3 \cdot d(f_c, c) + 4 \cdot d(c, s^c), \end{aligned}$$

which implies the second inequality from the statement of Lemma 3. The first one directly comes from the triangle inequality $d(c, f_c) \leq d(c, s^c) + d(s^c, s^{f_c}) + d(s^{f_c}, f_c) = d_\ell(c, f_c)$, completing the whole proof. ◀

The next lemma is quite simple. Its proof follows from the fact that metric d_ℓ dominates the metric d , i.e., $d_\ell(u, v) \geq d(u, v)$ for all pairs of vertices $u, v \in C \cup F$.

► **Lemma 6** (Going back from ℓ -centered metric d_ℓ to d). *Any solution for the ℓ -centered metric d_ℓ can be embedded back into d without any loss:*

$$\text{cost}(\phi^{ALG(d_\ell)}, d_\ell) \geq \text{cost}(\phi^{ALG(d_\ell)}, d).$$

Blending together Lemmas 3 and 6 we can state the following lemma about reducing the CKM problem to ℓ -centered instances.

► **Lemma 7.** *Suppose we are given a solution $ALG_{unc}^\ell(d)$ for the UNCAPACITATED KM problem on metric d which opens ℓ centers, but β -approximates the optimum solution $OPT_{unc}^k(d)$ for UNCAPACITATED KM problem with k centers, i.e., $cost(ALG_{unc}^\ell(d), d) \leq \beta \cdot cost(OPT_{unc}^k(d), d)$. Suppose we are given an α -approximation algorithm for the CKM problem on ℓ -centered instances. If so, then we can construct an $\alpha \cdot (3 + 4\beta)$ -approximation algorithm for CKM on general instances.*

Proof. Suppose that we have an α -approximation solution for the ℓ -centered instance with metric d_ℓ , i.e., $ALG(d_\ell)$ such that

$$cost(\phi^{ALG(d_\ell)}, d_\ell) \leq \alpha \cdot cost(\phi^{OPT(d_\ell)}, d_\ell).$$

Since $OPT(d)$ is some solution for the ℓ -centered instance with metric d_ℓ we have

$$cost(\phi^{ALG(d_\ell)}, d_\ell) \leq \alpha \cdot cost(\phi^{OPT(d_\ell)}, d_\ell) \leq \alpha \cdot cost(\phi^{OPT(d)}, d_\ell).$$

And from Lemma 3 we have that

$$\begin{aligned} cost(\phi^{ALG(d_\ell)}, d_\ell) &\leq \alpha \cdot cost(\phi^{OPT(d_\ell)}, d_\ell) \\ &\leq \alpha \cdot cost(\phi^{OPT(d)}, d_\ell) \\ &\leq \alpha \left(3 \cdot cost(\phi^{OPT(d)}, d) + 4 \cdot cost(\phi^{ALG_{unc}^\ell(d)}, d) \right). \end{aligned}$$

Since solution $ALG_{unc}^\ell(d)$ β -approximates the optimal solution $OPT_{unc}^k(d)$ for UNCAPACITATED KM with k centers on metric d , we have that

$$cost(\phi^{ALG_{unc}^\ell(d)}, d) \leq \beta \cdot cost(\phi^{OPT_{unc}^k(d)}, d) \leq \beta \cdot cost(\phi^{OPT(d)}, d).$$

The second inequality $cost(\phi^{OPT_{unc}^k(d)}, d) \leq cost(\phi^{OPT(d)}, d)$ follows from an obvious fact that uncapacitated version of the problem is easier than the capacitated. Hence

$$\begin{aligned} cost(\phi^{ALG(d_\ell)}, d_\ell) &\leq \alpha \left(3 \cdot cost(\phi^{OPT(d)}, d) + 4 \cdot cost(\phi^{ALG_{unc}^\ell(d)}, d) \right) \\ &\leq \alpha \left(3 \cdot cost(\phi^{OPT(d)}, d) + 4\beta \cdot cost(\phi^{OPT(d)}, d) \right) \\ &\leq \alpha (3 + 4\beta) \cdot cost(\phi^{OPT(d)}, d). \end{aligned}$$

Since without any loss we can embed the solution $ALG(d_\ell)$ for the ℓ -centered metric d_ℓ into the initial metric d (Lemma 6) we obtain an $\alpha \cdot (3 + 4\beta)$ -approximation algorithm. The claim follows. ◀

3 $\mathcal{O}(\log k)$ -approximation in polynomial time

In this section we present a folklore polynomial-time $\mathcal{O}(\log k)$ -approximation algorithm for CKM. Since constant-factor approximation algorithms for UNCAPACITATED KM exist [9], it is a clear consequence of Lemma 7 with β being constant that it is sufficient for us to construct an $\mathcal{O}(\log k)$ -approximation algorithm for the k -centered instances.

A standard tool to provide such a guarantee is the *Probabilistic Tree Embedding* by [15]. This makes our algorithm a randomized one, but if needed, it is possible to derandomize it using the ideas from [8].

► **Definition 8.** A set of metric spaces \mathcal{T} together with a probability distribution $\pi_{\mathcal{T}}$ over \mathcal{T} probabilistically α -approximates the metric space (X, d) if

1. Every metric $\tau \in \mathcal{T}$ dominates (X, d) , that is, $d(x, y) \leq \tau(x, y) \forall x, y \in X$.
2. For every pair of points $x, y \in X$ its expected distance is not expanded by more than α , i.e.,

$$\mathbb{E}_{\tau \sim \pi_{\mathcal{T}}}[\tau(x, y)] \leq \alpha \cdot d(x, y).$$

It is a well-known fact, that any metric (X, d) , can be probabilistically $\mathcal{O}(\log |X|)$ -approximated by a distribution of tree metrics, such that the points in X are the leaves in the resulting tree [15].

As described in Definition 2, our k -centered metric d_k is induced by a graph composed of two layers – the set S of k vertices connected in a clique, and the rest of vertices, $F \cup C$, each connected to only one vertex in S . Let T be a random tree embedding of the set S (with a metric function d_T). A modified instance G_T of our problem is created by replacing the clique S with its tree approximation T .

► **Lemma 9.** An optimum solution for CKM on the instance G_T is in expectation at most $\mathcal{O}(\log k)$ times larger than the optimum for the metric d_k .

Proof. $OPT(d_k)$ denotes the optimum mapping of clients to k facilities in the k -centered metric d_k . Consider client c and facility $f = \phi^{OPT(d_k)}(c)$. Let now s^c be the center of c and s^f the center of f . The cost of connecting client c to f amounts to

$$d_k(c, f) = d_k(c, s^c) + d_k(s^c, s^f) + d_k(s^f, f)$$

in the metric d_k .

The guarantee of tree embeddings gives us an upper bound on a cost of applying the same mapping in the instance G_T ,

$$\begin{aligned} \mathbb{E}[d_T(c, f)] &= d_k(c, s^c) + \mathbb{E}[d_T(s^c, s^f)] + d_k(s^f, f) \\ &\leq d_k(c, s^c) + \mathcal{O}(\log k) \cdot d_k(s^c, s^f) + d_k(s^f, f) \\ &\leq \mathcal{O}(\log k) \cdot d_k(c, f). \end{aligned}$$

Which means that $\mathbb{E}[\text{cost}(\phi^{OPT(d_k)}, d_T)] \leq \mathcal{O}(\log k) \cdot \text{cost}(\phi^{OPT(d_k)}, d_{G_k})$. Moreover, $OPT(d_k)$ might not be the optimal solution for the metric d_T , yet its optimal solution can only have smaller cost:

$$\text{cost}(\phi^{OPT(d_T)}, d_T) \leq \text{cost}(\phi^{OPT(d_k)}, d_T) \quad \blacktriangleleft$$

► **Theorem 10.** The CKM problem admits an $\mathcal{O}(\log k)$ -approximation algorithm with polynomial running time.

Proof. After applying the probabilistic tree embedding to the graph inducing d_k – as presented in Lemma 9 – we obtain a tree instance G_T . It should come as no surprise that the problem is polynomially solvable on trees and we explain how to find the optimum solution on G_T in Lemma 12. The assignment $\phi^{OPT(d_T)}$, which yields the minimum cost on the tree G_T , can be now used to match clients to facilities in the original instance. It does not incur any additional cost, as

$$\text{cost}(\phi^{OPT(d_T)}, d_T) \geq \text{cost}(\phi^{OPT(d_T)}, d_k) \geq \text{cost}(\phi^{OPT(d_T)}, d)$$

from the property (1) of Definition 8 and Lemma 6. Combining this with a bound on $\mathbb{E}[\text{cost}(\phi^{OPT(d_k)}, d_T)]$ from Lemma 9 finishes the proof. \blacktriangleleft

3.1 CKM on a tree

The second ingredient of the $\mathcal{O}(\log k)$ -approximation for CKM is an exact algorithm solving the problem on trees. We will now describe a simple, polynomial-time procedure for this special case. In our algorithm we can assume, that all the clients and facilities reside in leaves, but the principle is easy to extend to the general problem on trees. We first turn the tree into a complete binary tree by adding dummy vertices and edges of length 0 (which may double its size).

Suppose we have a subtree t of the tree instance, hanging on an edge e_t . Once we have decided, which facilities to open inside the subtree t , we know if their total capacity is sufficient to serve all the clients inside t . If not, then we need to route some clients' connections to the facilities outside through the edge e_t . However, if the facilities we have opened in t have enough total capacity to serve some b clients from the outside, we will connect them through the edge e_t .

► **Definition 11.** $D(t, k', b)$, for subtree t , number $k' \in \{0, \dots, k\}$ of facilities and balance $b \in \{-n, \dots, n\}$, is the minimum cost of opening exactly k' facilities in t and routing exactly b clients down through e_t ($b < 0$ would mean that we are routing $-b$ clients up). The cost of routing is counted to the top endpoint of e_t .

► **Lemma 12.** The CKM problem on trees admits a polynomial time exact algorithm.

Proof. Computing $D(t, k', b)$ on t with two children t_1 and t_2 amounts to finding k'_1, k'_2, b_1 and b_2 that minimize

$$D(t_1, k'_1, b_1) + D(t_2, k'_2, b_2),$$

such that $b_1 + b_2 = b$ and $k'_1 + k'_2 = k'$. They can be trivially found in $\mathcal{O}(k \cdot n)$ time for a single pair $\langle k', b \rangle$. Once k'_1, k'_2, b_1 and b_2 are found, we set

$$D(t, k', b) = D(t_1, k'_1, b_1) + D(t_2, k'_2, b_2) + d(e_t) \cdot |b|,$$

where $d(e)$ is the length of the edge in our tree. For a leaf l , $D(l, k', b)$ is defined naturally, depending on whether the leaf holds a client or a facility. Note, that for a leaf with a facility, $D(l, 1, b)$ is finite also for b smaller than the capacity of the facility, as the optimal solution might not use it entirely. Finally, the optimum solution to the CKM problem on the entire tree T is equal to $\min_{k' \in \{1, \dots, k\}} D(T, k', 0)$. ◀

4 Constant factor approximation

In this section we present the main result of the paper which is a $(7 + \varepsilon)$ -approximation algorithm for the NON-UNIFORM CKM problem. We precede it with a $(7 + \varepsilon)$ -approximation algorithm for the UNIFORM CKM problem to introduce the ideas gradually. Both algorithms enumerate configurations of open facilities' locations, and as a subroutine we need to use an algorithm which, for a fixed configuration of k open facilities, finds the optimal assignment of clients to facilities. This subroutine is presented in the following subsection.

4.1 Optimal mapping subroutine

We are given an ℓ -centered metric instance $(F \cup C \cup S, d_\ell)$ of the k -median problem. Suppose that we have already decided to open a fixed subset $F^{open} \subseteq F$ of the facilities assume $|F^{open}| \leq k$. and we look for a mapping $\phi : C \rightarrow F^{open}$. In the uncapacitated case we

can just assign each client to the closest facility in F^{open} . It turns out that even in the capacitated setting we can find the mapping ϕ optimally in polynomial time for a given F^{open} . We state the problem of finding the optimal ϕ as an integer program:

$$\begin{aligned}
 & \text{minimize} && \sum_{c \in C} \sum_{f \in F^{open}} d_\ell(c, f) \cdot x_{c,f} && \text{(MAPPING-IP)} \\
 & \text{subject to} && \sum_{f \in F^{open}} x_{c,f} = 1 && \forall c \in C, \\
 & && \sum_{c \in C} x_{c,f} \leq u_f && \forall f \in F^{open}, \\
 & && x_{c,f} \in \{0, 1\}.
 \end{aligned}$$

In the above program $x_{c,f} = 1$ represents the fact that $\phi(c) = f$.

► **Lemma 13.** *We can find an optimal solution to the (MAPPING-IP) in polynomial time.*

Proof. The proof follows from the fact that the relaxation of the above integer program – a program which differs from (MAPPING-IP) only with the $x_{c,f} \geq 0$ constraints instead of $x_{c,f} \in \{0, 1\}$ – has an optimal solution which is integral. To see this, observe that the linear program is a formulation of the transportation problem. For such a linear program, the constraint matrix is totally unimodular, which implies the integrality of an extremal solution. See [26] for a reference. ◀

4.2 Uniform case

We begin with a parameterized algorithm for the uniform case. It is simpler than the general case, as knowing the number of facilities to open in f -cluster $F(s)$ allows us to choose them greedily.

► **Lemma 14.** *UNIFORM CKM can be solved exactly in time $\ell^k \cdot n^{\mathcal{O}(1)}$ on ℓ -centered instances.*

Proof. Let $(F \cup C \cup S, d_\ell)$ be the ℓ -centered metric. Note that the f -clusters partition the whole set of facilities, i.e., $\cup_{s \in S} F(s) = F$. Let $OPT(d_\ell)$ be an optimal solution for the CKM problem on d_ℓ . Every facility $f \in F$ belongs to exactly one f -cluster $F(s)$. Hence, the f -clusters partition the set of k facilities opened by $OPT(d_\ell)$. Let us look at all the facilities from a particular f -cluster $F(s)$ opened by $OPT(d_\ell)$, and suppose that $OPT(d_\ell)$ opens k_s of facilities in $F(s)$. Since we consider a uniform capacity case, we can assume without loss that these k_s open facilities from $F(s)$ are exactly the ones that are closest to s .

Therefore, if we know what is the number of facilities that $OPT(d_\ell)$ opens in each f -cluster, then we would know what the exact set of open facilities in $OPT(d_\ell)$ is due to the greediness in each f -cluster. To find out this allocation we can simply enumerate over all possibilities. We just need to scan over all configurations $(k_s)_{s \in S}$ where $\sum_s k_s = k$. Since there are k facilities to open, and each of them can belong to one of ℓ f -clusters $F(s)$, there are at most ℓ^k possible configurations. Of course some configurations may not be feasible since it may happen that $k_s > |F(s)|$, but these can be simply ignored.

For each configuration $(k_s)_{s \in S}$ we need to find the optimal mapping of clients to the set of open facilities that preserves their capacities. Let $F((k_s)_{s \in S})$ be the set of open facilities induced by configuration $(k_s)_{s \in S}$, that is, where we greedily open k_s facilities in f -cluster $F(s)$. Given $F((k_s)_{s \in S})$, to find the optimal mapping we use the polynomial time exact algorithm from Lemma 13 with $F^{open} = F((k_s)_{s \in S})$.

Once we know the optimal assignment for each configuration, we can simply take the cheapest one, knowing that it is the optimal one. This proves the lemma. ◀

This lemma suffices to obtain a $(7 + \varepsilon)$ -approximation for UNIFORM CKM with a reasoning that we will present in Theorem 16 in full generality.

4.3 Non-uniform case

► **Lemma 15.** NON-UNIFORM CKM can be solved with approximation ratio $(1 + \varepsilon)$ in time $\mathcal{O}\left(\ell \cdot \frac{1}{\varepsilon} \ln \frac{n}{\varepsilon}\right)^k n^{\mathcal{O}(1)}$ on ℓ -centered instances.

Proof. We begin with guessing the largest distance in d_ℓ between a client and a facility that would appear in the optimal solution – let us denote this quantity as D . There are at most $\mathcal{O}(n^2)$ choices for D , and from now we assume that it is guessed correctly. Note that $D \leq \text{cost}(OPT(d_\ell), d_\ell)$ and $D \geq d(f, s_f)$ for all facilities opened by $OPT(d_\ell)$.

Consider the set of facilities $F(s)$ in the cluster of a center s . We can remove all facilities f such that $d(s, f) > D$, because they cannot be a part of the optimal solution. Let us partition remaining facilities from $F(s)$ into buckets $F_0(s), F_1(s), \dots, F_{\lceil \log_{1+\varepsilon} \frac{n}{\varepsilon} \rceil}(s)$, such that

$$F_i(s) = \begin{cases} \left\{ f \in F(s) \mid d(s, f) \in \left[(1 + \varepsilon)^{-(i+1)} D, (1 + \varepsilon)^{-i} D \right] \right\} & \text{for } i < \lceil \log_{1+\varepsilon} \frac{n}{\varepsilon} \rceil \\ \left\{ f \in F(s) \mid d(s, f) \in \left[0, (1 + \varepsilon)^{-\lceil \log_{1+\varepsilon} \frac{n}{\varepsilon} \rceil} D \right] \right\} & \text{for } i = \lceil \log_{1+\varepsilon} \frac{n}{\varepsilon} \rceil \end{cases}$$

The number of buckets equals $\log_{1+\varepsilon} \frac{n}{\varepsilon} = \frac{1}{\ln(1+\varepsilon)} \ln \frac{n}{\varepsilon} = \mathcal{O}\left(\frac{1}{\varepsilon} \ln \frac{n}{\varepsilon}\right)$. We modify the metric again by setting $d'_\ell(s, f) = (1 + \varepsilon)^{-i} D$ for $f \in F_i(s)$. The distances within S remain untouched. Observe that the distances can only increase.

We shall guess the structure of the solution $OPT(d'_\ell)$ similarly as in Lemma 14. For each of the k facilities, we can choose its location as follows: first we choose one of the ℓ -centers s (ℓ choices), and then we choose one of the $F_i(s)$ partitions ($\mathcal{O}\left(\frac{1}{\varepsilon} \ln \frac{n}{\varepsilon}\right)$ choices). Let us denote the number of facilities in a particular partition $F_i(s)$ as $k_{s,i}$. We can assume that $k_{s,i} \leq |F_i(s)|$ because otherwise we know that the guess was incorrect. Since $d'_\ell(s, f)$ is the same for all $f \in F_i(s)$, we can assume the optimal solution opens $k_{s,i}$ facilities with the biggest capacities.

Once we establish the set of facilities to open, we can find the optimal assignment in metric d'_ℓ using the polynomial time exact subroutine from Lemma 13.

The total time complexity of solving the problem exactly over d'_ℓ equals the running time of the subroutine times the number of possible configurations, which is $\mathcal{O}\left(\ell \cdot \frac{1}{\varepsilon} \ln \frac{n}{\varepsilon}\right)^k n^{\mathcal{O}(1)}$.

It remains to prove that the algorithm yields a proper approximation. We will show that for any solution SOL it holds that

$$\text{cost}(\phi^{SOL}, d_\ell) \leq \text{cost}(\phi^{SOL}, d'_\ell) \leq (1 + \varepsilon) \cdot \text{cost}(\phi^{SOL}, d_\ell) + \varepsilon \cdot D. \quad (1)$$

By substituting $SOL = OPT(d_\ell)$ we learn that there exists a solution over metric d'_ℓ of cost at most $(1 + \varepsilon) \cdot \text{cost}(\phi^{OPT(d_\ell)}, d_\ell) + \varepsilon \cdot D \leq (1 + 2\varepsilon) \cdot \text{cost}(\phi^{OPT(d_\ell)}, d_\ell)$ for correctly guessed D . Therefore the cost of the solution found by our algorithm cannot be larger. Finally we substitute this solution as SOL to see that its cost cannot increase when returning to metric d_ℓ . The claim will follow by adjusting ε .

The first inequality in (1) is straightforward because d'_ℓ dominates d_ℓ . Consider now a pair $(c, f = \phi^{SOL}(c))$, where $f \in F_i(s)$. If $i < \lceil \log_{1+\varepsilon} \frac{n}{\varepsilon} \rceil$, then $d_\ell(c, f) \leq d'_\ell(c, f) \leq (1 + \varepsilon) \cdot d_\ell(c, f)$, so the cost of connecting such pairs increases at most by a multiplicative factor $(1 + \varepsilon)$ during the metric switch. If $i = \lceil \log_{1+\varepsilon} \frac{n}{\varepsilon} \rceil$, then $d'_\ell(s, f) = \frac{\varepsilon D}{n}$. Since there are at most n such pairs, the total additive cost increase is bounded by $\varepsilon \cdot D$. ◀

► **Theorem 16.** NON-UNIFORM CKM can be solved with approximation ratio $(7 + \epsilon)$ in time $(k/\epsilon)^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$.

Proof. From Lemma 15 we know that we can get a $(1 + \epsilon)$ -approximation algorithm for the NON-UNIFORM CKM problem on ℓ -centered instances in time $(\mathcal{O}(\ell \cdot \frac{1}{\epsilon} \ln \frac{n}{\epsilon}))^k n^{\mathcal{O}(1)}$. We shall use the $(1 + \epsilon)$ -approximation for UNCAPACITATED KM by Lin and Vitter [24], that opens at most $\ell = (1 + \frac{1}{\epsilon}) k \cdot (\ln n + 1)$ facilities. By plugging this subroutine to find ℓ -centers into the Lemma 7 together with Lemma 15, we obtain a $(7 + \epsilon)$ -approximation algorithm for the general NON-UNIFORM CKM problem with running time

$$\mathcal{O}\left(\left(\left(1 + \frac{1}{\epsilon}\right) k \cdot (\ln n + 1) \cdot \frac{1}{\epsilon} \ln \frac{n}{\epsilon}\right)^k\right) n^{\mathcal{O}(1)} = \mathcal{O}\left(\left(\frac{1}{\epsilon^{\mathcal{O}(1)}} k \ln^2 n\right)^k\right) n^{\mathcal{O}(1)}.$$

Finally, we use standard arguments to show that $(\ln n)^{2k} \leq \max(n, k^{\mathcal{O}(k)})$. Consider two cases. If $\frac{\ln n}{2 \ln \ln n} \leq k$, then by inverting we know that $\ln n = \mathcal{O}(k \ln k)$, and so $(\ln n)^{2k} = k^{\mathcal{O}(k)}$. Suppose now that $\frac{\ln n}{2 \ln \ln n} > k$. In this case

$$(\ln n)^{2k} < (\ln n)^{\frac{\ln n}{\ln \ln n}} = e^{\ln \ln n \cdot \frac{\ln n}{\ln \ln n}} = n. \quad \blacktriangleleft$$

5 Conclusions and open problems

We have presented a $(7 + \epsilon)$ -approximation algorithm for the CKM problem, which consists of three building blocks: approximation for UNCAPACITATED KM, metric embedding into a simpler structure, and a parameterized algorithm working on ℓ -centered instances.

Whereas the first and the last ingredient are almost lossless from the approximation point of view, the embedding procedure seems to be the main bottleneck for obtaining a better approximation guarantee. One can imagine that a different technique would allow to obtain a $(1 + \epsilon)$ -approximation in FPT time. We believe that finding such an algorithm or ruling out its existence is an interesting research direction.

Another avenue for improvement is processing k -centered instances in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$. Such a routine would reduce the running time of the whole algorithm to single exponential. In order to do so, one could replace the subroutine for UNCAPACITATED KM by Lin and Vitter [24] with a standard approximation algorithm that opens exactly k facilities, what would moderately increase the constant in approximation ratio.

Finally, whereas we have used the framework of ℓ -centered instances to devise an FPT approximation, it might be possible to explore the structure of special instances further and find a polynomial time approximation algorithm. This could yield an improvement over the $\mathcal{O}(\log k)$ -approximation ratio for CKM, which remains a major open problem.

References

- 1 Karen Aardal, Pieter L van den Berg, Dion Gijswijt, and Shanfei Li. Approximation algorithms for hard capacitated k-facility location problems. *European Journal of Operational Research*, 242(2):358–368, 2015.
- 2 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.
- 3 Jarosław Byrka, Krzysztof Fleszar, Bartosz Rybicki, and Joachim Spoerhase. Bi-factor approximation algorithms for hard capacitated k-median problems. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 722–736. SIAM, 2015.

- 4 Jarosław Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median, and positive correlation in budgeted optimization. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 737–756. SIAM, 2015.
- 5 Jaroslaw Byrka, Bartosz Rybicki, and Sumedha Uniyal. An Approximation Algorithm for Uniform Capacitated k-Median Problem with $1 + \epsilon$ Capacity Violation. In *Integer Programming and Combinatorial Optimization - 18th International Conference, IPCO 2016, Liège, Belgium, June 1-3, 2016, Proceedings*, pages 262–274, 2016. doi:10.1007/978-3-319-33461-5_22.
- 6 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 *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, pages 743–754. IEEE, 2017.
- 7 Moses Charikar, Chandra Chekuri, Ashish Goel, and Sudipto Guha. Rounding via Trees: Deterministic Approximation Algorithms for Group Steiner Trees and k -Median. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 114–123, 1998. doi:10.1145/276698.276719.
- 8 Moses Charikar, Chandra Chekuri, Ashish Goel, Sudipto Guha, and Serge A. Plotkin. Approximating a Finite Metric by a Small Number of Tree Metrics. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 379–388, 1998. doi:10.1109/SFCS.1998.743488.
- 9 Moses Charikar, Sudipto Guha, Éva Tardos, and David B Shmoys. A constant-factor approximation algorithm for the k-median problem. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 1–10. ACM, 1999.
- 10 Julia Chuzhoy and Yuval Rabani. Approximating k-median with non-uniform capacities. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 952–958. Society for Industrial and Applied Mathematics, 2005.
- 11 Vincent Cohen-Addad and Jason Li. On the Fixed-Parameter Tractability of Capacitated Clustering. In *to appear in the 46th International Colloquium on Automata, Languages and Programming (ICALP)*, 2019.
- 12 Marek Cygan, MohammadTaghi Hajiaghayi, and Samir Khuller. LP rounding for k-centers with non-uniform hard capacities. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 273–282. IEEE, 2012.
- 13 Artur Czumaj and Christian Sohler. Small space representations for metric min-sum k-clustering and their applications. *Theory of Computing Systems*, 46(3):416–442, 2010.
- 14 H. Gökalp Demirci and Shi Li. Constant Approximation for Capacitated k-Median with $(1 + \epsilon)$ -Capacity Violation. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 73:1–73:14, 2016. doi:10.4230/LIPIcs.ICALP.2016.73.
- 15 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 448–455, 2003. doi:10.1145/780542.780608.
- 16 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-Deletion: Approximation, Kernelization and Optimal FPT Algorithms. In *Proceedings of the 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science, FOCS '12*, pages 470–479, Washington, DC, USA, 2012. IEEE Computer Society.
- 17 Anupam Gupta, Euiwoong Lee, and Jason Li. An FPT algorithm beating 2-approximation for k-cut. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2821–2837. Society for Industrial and Applied Mathematics, 2018.
- 18 Anupam Gupta, Euiwoong Lee, Jason Li, Pasin Manurangsi, and Michal Włodarczyk. Losing Treewidth by Separating Subsets. *CoRR*, abs/1804.01366, 2018. arXiv:1804.01366.

- 19 Madhukar R Korupolu, C Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. *Journal of algorithms*, 37(1):146–188, 2000.
- 20 Euiwoong Lee. Partitioning a graph into small pieces with applications to path transversal. *Mathematical Programming*, 2018. Preliminary version in SODA 2017.
- 21 Shi Li. On uniform capacitated k -median beyond the natural LP relaxation. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 696–707. SIAM, 2015.
- 22 Shi Li. Approximating capacitated k -median with $(1 + \epsilon)k$ open facilities. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, To Appear, 2016.
- 23 Shi Li and Ola Svensson. Approximating k -median via pseudo-approximation. In *proceedings of the forty-fifth annual ACM symposium on theory of computing*, pages 901–910. ACM, 2013.
- 24 Jyh-Han Lin and Jeffrey Scott Vitter. ϵ -Approximations with Minimum Packing Constraint Violation (Extended Abstract). In *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing, STOC '92*, pages 771–782, New York, NY, USA, 1992. ACM. doi:10.1145/129712.129787.
- 25 Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. On the Parameterized Complexity of Approximating Dominating Set. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 1283–1296, New York, NY, USA, 2018. ACM. doi:10.1145/3188745.3188896.
- 26 Alexander Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*. Springer, 2003.
- 27 Yicheng Xu, Yong Zhang, and Yifei Zou. A constant parameterized approximation for hard-capacitated k -means. *CoRR*, abs/1901.04628, 2019. arXiv:1901.04628.

Fragile Complexity of Comparison-Based Algorithms*

Peyman Afshani

Aarhus University, Denmark
peyman@cs.au.dk

Rolf Fagerberg

University of Southern Denmark, Odense, Denmark
rolf@imada.sdu.dk

David Hammer

Goethe University Frankfurt, Germany
University of Southern Denmark, Odense, Denmark
hammer@imada.sdu.dk

Riko Jacob

IT University of Copenhagen, Denmark
rikj@itu.dk

Irina Kostitsyna

TU Eindhoven, The Netherlands
i.kostitsyna@tue.nl

Ulrich Meyer

Goethe University Frankfurt, Germany
umeyer@ae.cs.uni-frankfurt.de

Manuel Penschuck

Goethe University Frankfurt, Germany
mpenschuck@ae.cs.uni-frankfurt.de

Nodari Sitchinava

University of Hawaii at Manoa
nodari@hawaii.edu

Abstract

We initiate a study of algorithms with a focus on the computational complexity of individual elements, and introduce the *fragile complexity* of comparison-based algorithms as the maximal number of comparisons any individual element takes part in. We give a number of upper and lower bounds on the fragile complexity for fundamental problems, including MINIMUM, SELECTION, SORTING and HEAP CONSTRUCTION. The results include both deterministic and randomized upper and lower bounds, and demonstrate a separation between the two settings for a number of problems. The depth of a comparator network is a straight-forward upper bound on the worst case fragile complexity of the corresponding fragile algorithm. We prove that fragile complexity is a different and strictly easier property than the depth of comparator networks, in the sense that for some problems a fragile complexity equal to the best network depth can be achieved with less total work and that with randomization, even a lower fragile complexity is possible.

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

Keywords and phrases Algorithms, comparison based algorithms, lower bounds

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.2

* This material is based upon work partially performed while attending AlgoPARC Workshop on Parallel Algorithms and Data Structures at the University of Hawaii at Manoa, in part supported by the National Science Foundation under Grant No. CCF-1745331.



© Peyman Afshani, Rolf Fagerberg, David Hammer, Riko Jacob, Irina Kostitsyna, Ulrich Meyer, Manuel Penschuck, and Nodari Sitchinava;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 2; pp. 2:1–2:19



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version <https://arxiv.org/abs/1901.02857>

Funding *Rolf Fagerberg*: Supported by the Independent Research Fund Denmark, Natural Sciences, grant DFF-7014-00041.

David Hammer: Supported by the Deutsche Forschungsgemeinschaft (DFG) under grants ME 2088/3-2 and ME 2088/4-2.

Ulrich Meyer: Supported by the Deutsche Forschungsgemeinschaft (DFG) under grants ME 2088/3-2 and ME 2088/4-2.

Manuel Penschuck: Supported by the Deutsche Forschungsgemeinschaft (DFG) under grants ME 2088/3-2 and ME 2088/4-2.

Nodari Sitchinava: Supported by the National Science Foundation under Grant No. CCF-1533823.

Acknowledgements We thank Steven Skiena for posing the original problem, and we thank Michael Bender, Rob Johnson, and Pat Morin for helpful discussions.

1 Introduction

Comparison-based algorithms is a classic and fundamental research area in computer science. Problems studied include minimum, median, sorting, searching, dictionaries, and priority queues, to name a few, and by now a huge body of work exists. The cost measure analyzed is almost always the total number of comparisons needed to solve the problem, either in the worst case or the expected case. Surprisingly, very little work has taken the viewpoint of the individual elements, asking the question: *how many comparisons must each element be subjected to?*

This question not only seems natural and theoretically fundamental, but is also practically well motivated: in many real world situations, comparisons involve some amount of destructive impact on the elements being compared, hence, keeping the maximum number of comparisons for each individual element low can be important. One example of such a situation is ranking of any type of consumable objects (wine, beer, food, produce), where each comparison reduces the available amount of the objects compared. Here, classical algorithms like QUICKSORT, which takes a single object and partitions the whole set with it, may use up this pivot element long before the algorithm completes. Another example is sports, where each comparison constitutes a match and takes a physical toll on the athletes involved. If a comparison scheme subjects one contestant to many more matches than others, both fairness to contestants and quality of result are impacted. The selection process could even contradict its own purpose – what is the use of finding a national boxing champion to represent a country at the Olympics if the person is injured in the process? Notice that in both examples above, quality of elements is difficult to measure objectively by a numerical value, hence one has to resort to relative ranking operations between opponents, i.e., comparisons. The detrimental impact of comparisons may also be of less directly physical nature, for instance if it involves a privacy risk for the elements compared, or if bias in the comparison process grows each time the same element is used.

► **Definition 1.** *We say that a comparison-based algorithm \mathcal{A} has fragile complexity $f(n)$ if each individual input element participates in at most $f(n)$ comparisons. We also say that \mathcal{A} has work $w(n)$ if it performs at most $w(n)$ comparisons in total. We say that a particular element e has fragile complexity $f_e(n)$ in \mathcal{A} if e participates in at most $f_e(n)$ comparisons.*

In this paper, we initiate the study of algorithms' fragile complexity – comparison-based complexity from the viewpoint of the individual elements – and present a number of upper and lower bounds on the fragile complexity for fundamental problems.

1.1 Previous work

One body of work relevant to what we study here is the study of sorting networks, propelled by the 1968 paper of Batcher [6]. In sorting networks, and more generally comparator networks, the notions of depth and size correspond to fragile complexity and standard worst case complexity,¹ respectively, since a network with depth $f(n)$ and size $w(n)$ easily can be converted into a comparison-based algorithm with fragile complexity $f(n)$ and work $w(n)$.

Batcher gave sorting networks with $\mathcal{O}(\log^2 n)$ depth and $\mathcal{O}(n \log^2 n)$ size, based on clever variants of the MERGESORT paradigm. A number of later constructions achieve the same bounds [10, 15, 16, 19], and for a long time it was an open question whether better results were possible. In the seminal result in 1983, Ajtai, Komlós, and Szemerédi [2, 3] answered this in the affirmative by constructing a sorting network of $\mathcal{O}(\log n)$ depth and $\mathcal{O}(n \log n)$ size. This construction is quite complex and involves expander graphs [21, 22], which can be viewed as objects encoding pseudorandomness, and which have many powerful applications in computer science and mathematics. The size of the constant factors in the asymptotic complexity of the AKS sorting network prevents it from being practical in any sense. It was later modified by others [8, 12, 17, 20], but finding a simple, optimal sorting network, in particular one not based on expander graphs, remains an open problem. Comparator networks for other problems, such as selection and heap construction have also been studied [5, 7, 14, 18, 23]. In all these problems the size of the network is super-linear.

As comparator networks of depth $f(n)$ and size $w(n)$ lead to comparison-based algorithms with $f(n)$ fragile complexity and $w(n)$ work, a natural question is, whether the two models are equivalent, or if there are problems for which comparison-based algorithms can achieve either asymptotically lower $f(n)$, or asymptotically lower $w(n)$ for the same $f(n)$.

One could also ask about the relationship between parallelism and fragile complexity. We note that parallel time in standard parallel models generally does not seem to capture fragile complexity. For example, even in the most restrictive exclusive read and exclusive write (EREW) PRAM model it is possible to create n copies of an element e in $\mathcal{O}(\log n)$ time and, thus, compare e to all the other input elements in $\mathcal{O}(\log n)$ time, resulting in $\mathcal{O}(\log n)$ parallel time but $\Omega(n)$ fragile complexity. Consequently, it is not clear whether Richard Cole's celebrated parallel merge sort algorithm [9] yields a comparison-based algorithm with low fragile complexity as it copies some elements.

1.2 Our contribution

In this paper we present algorithms and lower bounds for a number of classical problems, summarized in Table 1. In particular, we study finding the MINIMUM (Section 2), the SELECTION problem (Section 3), and SORTING (Section 4).

Minimum. The case of the deterministic algorithms is clear: using an adversary lower bound, we show that the minimum element needs to suffer $\Omega(\log n)$ comparisons and a tournament tree trivially achieves this bound (Subsection 2.1). The randomized case, however, is much more interesting. We obtain a simple algorithm where the probability of the minimum element suffering k comparisons is doubly exponentially low in k , roughly $1/2^{2^k}$ (see Subsection 2.2). As a result, the $\Theta(\log n)$ deterministic fragile complexity can be lowered to $\mathcal{O}(1)$ expected or even $\mathcal{O}(\log \log n)$ with high probability. We also show this latter high probability case is lower

¹ For clarity, in the rest of the paper we call standard worst case complexity *work*.

2:4 Fragile Complexity of Comparison-Based Algorithms

■ **Table 1** Summary of presented results. Notation: $f(n)$ means fragile complexity; $w(n)$ means work; $\langle f_m(n), f_{rem}(n) \rangle$ means fragile complexity for the selected element (minimum/median) and for the remaining elements, respectively – except for lower bounds, where it means \langle expected for the selected, limit for remaining \rangle ; \dagger means holds in expectation; \ddagger means holds with high probability $(1 - 1/n)$. $\varepsilon > 0$ is an arbitrary constant.

Problem		Upper		Lower
		$f(n)$	$w(n)$	$f(n)$
MINIMUM	Determ. (Sec. 2)	$\mathcal{O}(\log n)$ [T 2]	$\mathcal{O}(n)$	$f_{\min} = \Omega(\log n)$ [T 2]
	Rand. (Sec. 2)	$\langle \mathcal{O}(\log_{\Delta} n)^{\dagger}, \mathcal{O}(\Delta + \log_{\Delta} n)^{\dagger} \rangle$ [T 9] $\langle \mathcal{O}(1)^{\dagger}, \mathcal{O}(n^{\varepsilon}) \rangle$ (setting $\Delta = n^{\varepsilon}$) $\mathcal{O}\left(\frac{\log n}{\log \log n}\right)^{\dagger}$ [Cor 11]	$\mathcal{O}(n)$	$\langle \Omega(\log_{\Delta} n)^{\dagger}, \Delta \rangle$ [T 10] $\Omega\left(\frac{\log n}{\log \log n}\right)^{\dagger}$ [Cor 11]
		$\langle \mathcal{O}(\log_{\Delta} n \log \log \Delta)^{\ddagger}, \mathcal{O}(\Delta + \log_{\Delta} n \log \log \Delta)^{\ddagger} \rangle$ [T 9]	$\mathcal{O}(n)$ $\mathcal{O}(n)$	$f_{\min} =$ $= \Omega(\log \log n)^{\ddagger}$ [T 14]
SELECTION	Determ. (Sec. 3)	$\mathcal{O}(\log n)$ [T 15]	$\mathcal{O}(n)$ [T 15]	$\Omega(\log n)$ [Cor 3]
	Rand. (Sec. 3)	$\langle \mathcal{O}(\log \log n)^{\dagger}, \mathcal{O}(\sqrt{n})^{\dagger} \rangle$ [T 17] $\langle \mathcal{O}\left(\frac{\log n}{\log \log n}\right)^{\dagger}, \mathcal{O}(\log^2 n)^{\dagger} \rangle$ [T 17]	$\mathcal{O}(n)^{\dagger}$	$\langle \Omega(\log_{\Delta} n)^{\dagger}, \Delta \rangle$ [T 10]
MERGE		Determ. (Sec. 4)	$\mathcal{O}(\log n)$ [T 24]	$\mathcal{O}(n)$
HEAP CONSTR.	Determ. (Sec. 5)	$\mathcal{O}(\log n)$ [Obs 2]	$\mathcal{O}(n)$	$\Omega(\log n)$ [T 2]

bounded by $\Omega(\log \log n)$ (Subsection 2.3). Furthermore, we can achieve a trade-off between the fragile complexity of the minimum element and the other elements. Here $\Delta = \Delta(n)$ is a parameter we can choose freely that basically upper bounds the fragile complexity of the non-minimal elements. We can find the minimum with $\mathcal{O}(\log_{\Delta} n)$ expected fragile complexity while all the other elements suffer $\mathcal{O}(\Delta + \log_{\Delta} n)$ comparisons (Subsection 2.3). Furthermore, this is tight: we show an $\Omega(\log_{\Delta} n)$ lower bound for the expected fragile complexity of the minimum element where the maximum fragile complexity of non-minimum elements is at most Δ .

Selection. Minimum finding is a special case of the selection problem where we are interested in finding an element of a given rank. As a result, all of our lower bounds apply to this problem as well. Regarding upper bounds, the deterministic case is trivial if we allow for $\mathcal{O}(n \log n)$ work (via sorting). We show that this can be reduced to $\mathcal{O}(n)$ time while keeping the fragile complexity of all the elements at $\mathcal{O}(\log n)$ (Section 3). Once again, randomization offers a substantial improvement: e.g., we can find the median in $\mathcal{O}(n)$ expected work and with $\mathcal{O}(\log \log n)$ expected fragile complexity while non-median elements suffer $\mathcal{O}(\sqrt{n})$ expected comparisons, or we can find the median in $\mathcal{O}(n)$ expected work and with $\mathcal{O}(\log n / \log \log n)$ expected fragile complexity while non-median elements suffer $\mathcal{O}(\log^2 n)$ expected comparisons.

Sorting and other results. The deterministic selection, sorting, and heap construction fragile complexities follow directly from the classical results in comparator networks [3, 7]. However, we show a separation between comparator networks and comparison-based algorithms for the problem of MEDIAN (Section 3) and HEAP CONSTRUCTION (Section 5), in the sense that depth/fragile complexity of $\mathcal{O}(\log n)$ can be achieved in $\mathcal{O}(n)$ work for comparison-based algorithms, but requires $\Omega(n \log n)$ [5] and $\Omega(n \log \log n)$ [7] sizes for comparator networks

for the two problems, respectively. For sorting the two models achieve the same complexities: $\mathcal{O}(\log n)$ depth/fragile complexity and $\mathcal{O}(n \log n)$ size/work, which are the optimal bounds in both models due to the $\Omega(\log n)$ lower bound on fragile complexity for MINIMUM (Theorem 2) and the standard $\Omega(n \log n)$ lower bound on work for comparison-based sorting. However, it is an open problem whether these bounds can be achieved by simpler sorting algorithms than sorting networks, in particular whether expander graphs are necessary. One intriguing conjecture could be that any comparison-based sorting algorithm with $\mathcal{O}(\log n)$ fragile complexity and $\mathcal{O}(n \log n)$ work implies an expander graph. This would imply expanders, optimal sorting networks and fragile-optimal comparison-based sorting algorithms to be equivalent, in the sense that they all encode the same level of pseudorandomness.

We note that our lower bound of $\Omega(\log^2 n)$ on the fragile complexity of MERGESORT (Theorem 19) implies the same lower bound on the depth of any sorting network based on binary merging, which explains why many of the existing simple sorting networks have $\Theta(\log^2 n)$ depth. Finally, our analysis of MERGESORT on random inputs (Theorem 23) shows a separation between deterministic and randomized fragile complexity for such algorithms. In summary, we consider the main contributions of this paper to be:

- the introduction of the model of fragile complexity, which we find intrinsically interesting, practically relevant, and surprisingly overlooked
- the separations between this model and the model of comparator networks
- the separations between the deterministic and randomized setting within the model
- the lower bounds on randomized minimum finding

Due to space constraints, some proofs only appear in the full paper [1].

2 Finding the minimum

2.1 Deterministic Algorithms

As a starting point, we study deterministic algorithms that find the minimum among an input of n elements. Our results here are simple but they act as interesting points of comparison against the subsequent non-trivial results on randomized algorithms.

► **Theorem 2.** *The fragile complexity of finding the minimum of n elements is $\lceil \log n \rceil$.*

Proof. The upper bound is achieved using a perfectly balanced tournament tree. The lower bound follows from a standard adversary argument. ◀

Observe that in addition to returning the minimum, the balanced tournament tree can also return the second smallest element, without any increase to the fragile complexity of the minimum. We refer to this deterministic algorithm that returns the smallest and the second smallest element of a set X as $\text{TournamentMinimum}(X)$.

► **Corollary 3.** *For any deterministic algorithm \mathcal{A} that finds the median of n elements, the fragile complexity of the median element is at least $\lceil \log n \rceil - 1$.*

Proof. By a standard padding argument with $n - 1$ small elements. ◀

2.2 Randomized Algorithms for Finding the Minimum

We now show that finding the minimum is provably easier for randomized algorithms than for deterministic algorithms. We define f_{\min} as the fragile complexity of the minimum and f_{rem} as the maximum fragile complexity of the remaining elements. For deterministic

2:6 Fragile Complexity of Comparison-Based Algorithms

algorithms we have shown that $f_{\min} \geq \log n$ regardless of f_{rem} . This is very different in the randomized setting. In particular, we first show that we can achieve $\mathbb{E}[f_{\min}] = O(1)$ and $f_{\min} = O(1) + \log \log n$ with high probability (we later show this high probability bound is also tight, Theorem 14).

```

1: procedure SAMPLEMINIMUM( $X$ )  $\triangleright$  Returns the smallest and 2nd smallest element of  $X$ 
2:   if  $|X| \leq 8$  return TOURNAMENTMINIMUM( $X$ )
3:   Let  $A \subset X$  be a uniform random sample of  $X$ , with  $|A| = \lceil |X|/2 \rceil$ 
4:   Let  $B \subset A$  be a uniform random sample of  $A$ , with  $|B| = \lfloor |X|^{2/3} \rfloor$ 
5:    $\triangleright$  The minimum is either in (i)  $C \subseteq X \setminus A$ , (ii)  $D \subseteq A \setminus B$  or (iii)  $B$ 
6:    $(b_1, b_2) = \text{SAMPLEMINIMUM}(B)$   $\triangleright$  the minimum participates only in case (iii)
7:   Let  $D = \{x \in A \setminus B \mid x < b_2\}$   $\triangleright$  the minimum is compared once only in case (ii)
8:   Let  $(a'_1, a'_2) = \text{SAMPLEMINIMUM}(D)$   $\triangleright$  only case (ii)
9:   Let  $(a_1, a_2) = \text{TOURNAMENTMINIMUM}(a'_1, a'_2, b_1, b_2)$   $\triangleright$  case (ii) and (iii)
10:  Let  $C = \{x \in X \setminus A \mid x < a_2\}$   $\triangleright$  only case (i)
11:  Let  $(c_1, c_2) = \text{TOURNAMENTMINIMUM}(C)$   $\triangleright$  only case (i)
12:  return TOURNAMENTMINIMUM( $a_1, a_2, c_1, c_2$ )  $\triangleright$  always

```

First, we show that this algorithm can actually find the minimum with expected constant number of comparisons. Later, we show that the probability that this algorithm performs t comparisons on the minimum drops roughly *doubly exponentially* on t .

We start with the simple worst-case analysis.

► **Lemma 4.** *Algorithm SAMPLEMINIMUM(X) achieves $f_{\min} \leq 3 \log |X|$ in the worst case.*

Proof. First, observe that the smallest element in Lines 9 and 12 participates in at most one comparison because pairs of elements are already sorted. Then the fragile complexity of the minimum is defined by the maximum of the three cases:

- (i) One comparison each in Lines 10 and 12, plus (by Theorem 2) $\lceil \log |C| \rceil \leq \log |X|$ comparisons in Line 11.
- (ii) One comparison each in Lines 7, 9, and 12, plus the recursive call in line 8.
- (iii) One comparison each in Lines 6, 9, and 12, plus the recursive call in line 6.

The recursive calls in lines 8 and 6 are on at most $|X|/2$ elements because $B \subset A$, $D \subset A$, and $|A| = \lceil |X|/2 \rceil$. Consequently, the fragile complexity of the minimum is defined by the recurrence

$$T(n) \leq \begin{cases} \max\{3 + T(n/2), 2 + \log n\} & \text{if } n > 8 \\ 3 & \text{if } n \leq 8 \end{cases},$$

which solves to $T(n) \leq 3 \log n$. ◀

► **Lemma 5.** *Assume that in Algorithm SAMPLEMINIMUM, the minimum y is in $X \setminus A$, i.e. we are in case (i). Then $\Pr[|C| = k \mid y \notin A] \leq \frac{k}{2^k}$ for any $k \geq 1$ and $n \geq 7$.*

Proof. There are $\binom{n-1}{\lceil n/2 \rceil}$ possible events of choosing a random subset $A \subset X$ of size $\lceil n/2 \rceil$ s.t. $y \notin A$. Let us count the number of the events $\{|C| = k \mid y \notin A\}$, which is equivalent to a_2 , the second smallest element of A , being larger than exactly $k + 1$ elements of X .

For simplicity of exposition, consider the elements of $X = \{x_1, \dots, x_n\}$ in sorted order. The minimum $y = x_1 \notin A$, therefore, a_1 (the smallest element of A) must be one of the k elements $\{x_2, \dots, x_{k+1}\}$. By the above observation, $a_2 = x_{k+2}$. And the remaining $\lceil n/2 \rceil - 2$ elements of A are chosen from among $\{x_{k+3}, \dots, x_n\}$. Therefore,

$$\Pr[|C| = k \mid y \notin A] = \frac{k \cdot \binom{n-(k+2)}{\lceil n/2 \rceil - 2}}{\binom{n-1}{\lceil n/2 \rceil}} = k \cdot \frac{(n-(k+2))!}{(\lceil n/2 \rceil - k)! (\lceil n/2 \rceil - 2)!} \cdot \frac{(\lceil n/2 \rceil)! (\lceil n/2 \rceil - 1)!}{(n-1)!}$$

Rearranging the terms, we get:

$$\Pr[|C| = k \mid y \notin A] = k \cdot \frac{(n-(k+2))!}{(n-1)!} \cdot \frac{(\lceil n/2 \rceil)!}{(\lceil n/2 \rceil - 2)!} \cdot \frac{(\lceil n/2 \rceil - 1)!}{(\lceil n/2 \rceil - k)!}$$

There are two cases to consider:

$$\begin{aligned} k = 1 : \quad \Pr[|C| = k \mid y \notin A] &= 1 \cdot \frac{1}{(n-1)(n-2)} \cdot \lceil n/2 \rceil (\lceil n/2 \rceil - 1) \cdot 1 \\ &\leq \frac{1}{(n-1)(n-2)} \cdot \frac{(n+1)}{2} \cdot \frac{(n-1)}{2} \\ &= \frac{n+1}{4 \cdot (n-2)} \leq \frac{1}{2} = \frac{k}{2^k} \quad \text{for every } n \geq 5. \end{aligned}$$

$$\begin{aligned} k \geq 2 : \quad \Pr[|C| = k \mid y \notin A] &= k \cdot \frac{1}{\prod_{i=1}^{k+1} (n-i)} \cdot \lceil n/2 \rceil (\lceil n/2 \rceil - 1) \cdot \prod_{i=1}^{k-1} \left(\left\lfloor \frac{n}{2} \right\rfloor - i \right) \\ &\leq k \cdot \frac{1}{\prod_{i=1}^{k+1} (n-i)} \cdot \frac{n+1}{2} \cdot \frac{n-1}{2} \cdot \prod_{i=1}^{k-1} \frac{n-2i}{2} \\ &\leq \frac{k}{2^{k+1}} \cdot (n+1)(n-1) \cdot \frac{\prod_{i=1}^{k-1} (n-2i)}{\prod_{i=1}^{k+1} (n-i)} \\ &\leq \frac{k}{2^{k+1}} \cdot (n+1)(n-1) \cdot \frac{n-2}{(n-1)(n-2)(n-3)} \\ &= \frac{k}{2^{k+1}} \cdot \frac{n+1}{n-3} \leq \frac{k}{2^{k+1}} \cdot 2 = \frac{k}{2^k} \quad \text{for every } n \geq 7. \quad \blacktriangleleft \end{aligned}$$

► **Theorem 6.** *Algorithm SAMPLEMINIMUM achieves $\mathbb{E}[f_{\min}] \leq 9$.*

Proof. By induction on the size of X . In the base case $|X| \leq 8$, clearly $f_{\min} \leq 3$, implying the theorem.

Now assume that the calls in Line 8 and Line 6 have the property that $\mathbb{E}[f(b_1)] \leq 9$ and $\mathbb{E}[f(a'_1)] \leq 9$. Both in case (ii) and case (iii), the expected number of comparisons of the minimum is $\leq 9 + 3$. Case (i) happens with probability at least $1/2$. In this case, the expected number of comparisons is 2 plus the ones from Line 11. By Lemma 5 we have $\Pr[|C| = k \mid \text{case (i)}] \leq k2^{-k}$. Because TOURNAMENTMINIMUM (actually any algorithm not repeating the same comparison) uses the minimum at most $k-1$ times, the expected number of comparisons in Line 11 is $\sum_{k=1}^{\lceil n/2 \rceil} (k-1)k2^{-k} \leq \sum_{k=1}^{\infty} (k-1)k2^{-k} \leq 4$. Combining the bounds we get $\mathbb{E}[f_{\min}] \leq \frac{9+3}{2} + \frac{2+4}{2} = 9$. ◀

Observe that the above proof did not use anything about the sampling of B , and also did not rely on TOURNAMENTMINIMUM.

► **Lemma 7.** *For $|X| > 2$ and any $\gamma > 1$: $\Pr[|D| \geq \gamma|X|^{1/3}] < |X| \exp(-\Theta(\gamma))$*

Proof. Let $n = |X|$, $a = |A| = \lceil n/2 \rceil$ and $b = |B| = \lfloor n^{2/3} \rfloor$. The construction of the set B can be viewed as the following experiment. Consider drawing without replacement from an urn with b blue and $a - b$ red marbles. The i -th smallest element of A is chosen into B iff the i -th draw from the urn results in a blue marble. Then $|D| \geq \gamma|X|^{1/3} = \gamma n^{1/3}$ implies that this experiment results in at most one blue marble among the first $t = \gamma n^{1/3}$ draws. There are precisely $t + 1$ elementary events that make up the condition $|D| \geq t$, namely that the i -th draw is a blue marble, and where $i = 0$ stands for the event “all t marbles are red”. Let us denote the probabilities of these elementary events as p_i .

Observe that each p_i can be expressed as a product of t factors, at least $t - 1$ of which stand for drawing a red marble, each upper bounded by $1 - \frac{b-1}{a}$. The remaining factor stands for drawing the first blue marble (from the urn with $a - i$ marbles, b of which are blue), or another red marble. In any case we can bound

$$p_i \leq \left(1 - \frac{b-1}{a}\right)^{t-1} \leq \left(1 - \frac{b-1}{a}\right)^{\gamma n^{1/3} - 1} = \exp\left(-\Theta\left(\frac{b\gamma n^{1/3}}{a}\right)\right).$$

Summing the $t + 1$ terms, and observing $t + 1 < n$ if the event can happen at all, we get

$$\Pr[|D| \geq \gamma|X|^{1/3}] < n \cdot \exp\left(-\Theta\left(\frac{\gamma n^{1/3} n^{2/3}}{n/2}\right)\right) = n \cdot \exp(-\Theta(\gamma)). \quad \blacktriangleleft$$

► **Theorem 8.** *There is a positive constant c , such that for any parameter $t \geq c$, the minimum in the Algorithm `SAMPLEMINIMUM`(X) participates in at most $O(t + \log \log |X|)$ comparisons with probability at least $1 - \exp(-2^t)2 \log \log |X|$.*

Proof. Let $n = |X|$ and y be the minimum element. In each recursion step, we have one of three cases: (i) $y \in C \subseteq X \setminus A$, (ii) $y \in D \subseteq A \setminus B$ or (iii) $y \in B$. Since the three sets are disjoint, the minimum always participates in at most one recursive call. Tracing only the recursive calls that include the minimum, we use the superscript $X^{(i)}$, $A^{(i)}$, $B^{(i)}$, $C^{(i)}$, and $D^{(i)}$ to denote these sets at depth i of the recursion.

Let h be the first recursive level when $y \in C^{(h)}$, i.e., $y \notin A^{(h)}$. It follows that y will not be involved in the future recursive calls because it is in a single call to `TOURNAMENTMINIMUM`. Thus, at this level of recursion, the number of comparisons that y will accumulate is equal to $O(1) + \log |C^{(h)}|$. To bound this quantity, let $k = 4 \cdot 2^t$. Then, by Lemma 5, $\Pr[|C^{(h)}| > k] \leq k2^{-k} = 4 \cdot 2^t \cdot 2^{-4 \cdot 2^t} = 4 \cdot 2^t \cdot 4^{-2^t} \cdot 4^{-2^t}$. Since $4x4^{-x} \leq 1$ for any $x \geq 1$, $\Pr[|C^{(h)}| > k] \leq 4^{-2^t}$ for any $t \geq 0$. I.e., the number of comparisons that y participates in at level h is at most $O(1) + \log k = O(1) + t$ with probability at least $1 - 4^{-2^t} \geq 1 - \exp(-2^t)$.

Thus, it remains to bound the number of comparisons involving y at the recursive levels $i \in [1, h - 1]$. In each of these recursive levels $y \notin C^{(i)}$, which only leaves the two cases: (ii) $y \in D^{(i)} \subseteq A^{(i)} \setminus B^{(i)}$ and (iii) $y \in B^{(i)}$. The element y is involved in at most $O(1)$ comparisons in lines 7, 9 and 12. The two remaining lines of the algorithm are lines 6 and 8 which are the recursive calls. We differentiate two types of recursive calls:

- Type 1: $|X^{(i)}| \leq 2^{4t}$. In this case, by Lemma 4, the algorithm will perform $O(t)$ comparisons at the recursive level i , as well as any subsequent recursive levels.
- Type 2: $|X^{(i)}| > 2^{4t}$. In this case, by Lemma 7 on the set $X^{(i)}$ and $\gamma = |X^{(i)}|^{1/3}$ we get:

$$\Pr[|D^{(i)}| \geq \gamma|X^{(i)}|^{1/3}] < |X^{(i)}| \exp\left(-\Theta\left(|X^{(i)}|^{1/3}\right)\right) < \exp\left(-\Theta\left(|X^{(i)}|^{1/3}\right)\right)$$

Note that since $|X^{(i)}|^{1/3} > 2^t$, by the definition of the Θ -notation, there exists a positive constant c , such that $\exp\left(-\Theta\left(|X^{(i)}|^{1/3}\right)\right) < \exp(-2^t)$. Thus, it follows that with probability $1 - \exp(-2^t)$, we will recurse on a subproblem of size at most $\gamma|X^{(i)}|^{1/3} \leq |X^{(i)}|^{2/3}$. Let G_i be this (good) event, and thus $\Pr[G_i] \geq 1 - \exp(-2^t)$.

Observe that the maximum number of times we can have good events of type 2 is very limited. With every such good event, the size of the subproblem decreases significantly and thus eventually we will arrive at a recursive call of type 1. Let j be this maximum number of “good” recursive levels of type 2. The problem size at the j -th such recursive level is at most $n^{(2/3)^{j-1}}$ and we must have that $n^{(2/3)^{j-1}} > 2^{4t}$ which reveals that we must have $j = O(\log \log n)$.

We are now almost done and we just need to use a union bound. Let G be the event that at the recursive level h , we perform at most $O(1) + t$ comparisons, and all the recursive levels of type 2 are good. G is the conjunction of at most $j + 1$ events and as we have shown, each such event holds with probability at least $1 - \exp(-2^t)$. Thus, it follows that G happens with probability $1 - (j + 1) \exp(-2^t) > 1 - 2 \log \log n \exp(-2^t)$. Furthermore, our arguments show that if G happens, then the minimum will only participate in $O(t + j) = O(t + \log \log n)$ comparisons. \blacktriangleleft

The major strengths of the above algorithm is the doubly exponential drop in probability of comparing the minimum with too many elements. Based on it, we can design another simple algorithm to provide a smooth trade-off between f_{\min} and f_{rem} . Let $2 \leq \Delta \leq n$ be an integral parameter. We will design an algorithm that achieves $\mathbb{E}[f_{\min}] = O(\log_{\Delta} n)$ and $f_{\min} = O(\log_{\Delta} n \cdot \log \log \Delta)$ whp, and $f_{\text{rem}} = \Delta + O(\log_{\Delta} n \cdot \log \log \Delta)$ whp. For simplicity we assume n is a power of Δ . We build a fixed tournament tree T of degree Δ and of height $\log_{\Delta} n$ on X . For a node $v \in T$, let $X(v)$ be the set of values in the subtree rooted at v . The following code computes $m(v)$, the minimum value of $X(v)$, for every node v .

```

1: procedure TREEMINIMUM $_{\Delta}(X)$ 
2:   For every leaf  $v$ , set  $m(v)$  equal to the single element of  $X(v)$ .
3:   For every internal node  $v$  with  $\Delta$  children  $u_1, \dots, u_{\Delta}$  where the values
       $m(u_1), \dots, m(u_{\Delta})$  are known, compute  $m(v)$  using SIMPLEMINIMUM algorithm on input
       $\{m(u_1), \dots, m(u_{\Delta})\}$ .
4:   Repeat the above step until the minimum of  $X$  is computed.

```

The correctness of TREEMINIMUM $_{\Delta}$ is trivial. So it remains to analyze its fragile complexity.

► Theorem 9. *In TREEMINIMUM $_{\Delta}$, $\mathbb{E}[f_{\min}] = O(\log_{\Delta} n)$ and $\mathbb{E}[f_{\text{rem}}] = \Delta + O(\log_{\Delta} n)$. Furthermore, with high probability, $f_{\min} = O\left(\frac{\log n \log \log \Delta}{\log \Delta}\right)$ and $f_{\text{rem}} = O\left(\Delta + \frac{\log n \log \log \Delta}{\log \Delta}\right)$.*

Proof. First, observe that $\mathbb{E}[f_{\min}] = O(\log_{\Delta} n)$ is an easy consequence of Theorem 6. Now we focus on high probability bounds. Let $k = c \cdot h \log \ln \Delta$, and $h = \log_{\Delta} n$ for a large enough constant c . There are h levels in T . Let \mathbf{f}_i be the random variable that counts the number of comparisons the minimum participates in at level i of T . Observe that these are independent random variables. Let f_1, \dots, f_h be integers such that $f_i \geq 1$ and $\sum_{i=1}^h f_i = k$, and let c' be the constant hidden in the big- O notation of Theorem 8. Use Theorem 8 h times (with n set to Δ , and $t = f_i$), and also bound $2 \log \log \Delta < \Delta$ to get

$$\Pr \left[\mathbf{f}_1 \geq c'(f_1 + \log \log \Delta) \vee \dots \vee \mathbf{f}_h \geq c'(f_h + \log \log \Delta) \right] \leq \Delta^h e^{-\sum_i 2^{f_i}} \leq \Delta^h e^{-h 2^{k/h}}$$

where the last inequality follows from the inequality of arithmetic and geometric means (specifically, observe that $\sum_{i=1}^h 2^{f_i}$ is minimized when all f_i 's are distributed evenly).

2:10 Fragile Complexity of Comparison-Based Algorithms

Now observe that the total number of different integral sequences f_1, \dots, f_h that sum up to k is bounded by $\binom{h+k}{h}$ (this is the classical problem of distributing k identical balls into h distinct bins). Thus, we have

$$\begin{aligned} \Pr[f_{\min} = O(k + h \log \log \Delta)] &\leq \binom{h+k}{h} \cdot \Delta^h \frac{1}{e^{h \cdot 2^{k/h}}} \leq \left(\frac{e(h+k)}{h}\right)^h \cdot \Delta^h \frac{1}{e^{h \cdot 2^{k/h}}} \\ &\leq \left(\frac{O\left(\frac{k}{h}\right) \cdot \Delta}{e^{2^{k/h}}}\right)^h = \left(\frac{O(\Delta^2)}{e^{2^c \log \ln \Delta}}\right)^h < \left(\frac{O(\Delta^2)}{e^{\ln^c \Delta}}\right)^h < \left(\frac{\Delta^3}{\Delta^{\ln^{c-1} \Delta}}\right)^h < \Delta^{-ch} = n^{-c} \end{aligned}$$

where in the last step we bound $(\ln \Delta)^{c-1} - 3 > c$ for large enough c and $\Delta \geq 3$. This is a high probability bound for f_{\min} . To bound f_{rem} , observe that for every non-minimum element x , there exists a lowest node v such that x is not $m(v)$. If x is not passed to the ancestors of v , x suffers at most Δ comparisons in v , and below v x behaves like the minimum element, which means that the above analysis applies. This yields that whp we have $f_{\text{rem}} = \Delta + O\left(\frac{\log n \log \log \Delta}{\log \Delta}\right)$. ◀

2.3 Randomized Lower Bounds for Finding the Minimum

2.3.1 Expected Lower Bound for the Fragile Complexity of the Minimum.

The following theorem is our main result.

► **Theorem 10.** *In any randomized minimum finding algorithm with fragile complexity of at most Δ for any element, the expected fragile complexity of the minimum is at least $\Omega(\log \Delta n)$.*

Note that this theorem implies the fragile complexity of finding the minimum:

► **Corollary 11.** *Let $f(n)$ be the expected fragile complexity of finding the minimum (i.e. the smallest function such that some algorithm achieves $f(n)$ fragile complexity for all elements (minimum and the rest) in expectation). Then $f(n) = \Theta\left(\frac{\log n}{\log \log n}\right)$.*

Proof. Use Theorem 9 as the upper bound and Theorem 10, both with $\Delta = \frac{\log n}{\log \log n}$, observing that if $f(n)$ is an upper bound that holds with high probability, it is also an upper bound on the expectation. ◀

To prove Theorem 10 we give a lower bound for a *deterministic* algorithm \mathcal{A} on a random input of n values, x_1, \dots, x_n where each x_i is chosen iid and uniformly in $(0, 1)$. By Yao's minimax principle, the lower bound on the expected fragile complexity of the minimum when running \mathcal{A} also holds for any randomized algorithm.

We prove our lower bound in a model that we call “comparisons with additional information (CAI)”: if the algorithm \mathcal{A} compares two elements x_i and x_j and it turns out that $x_i < x_j$, then the value x_j is revealed to the algorithm. Clearly, the algorithm can only do better with this extra information. The heart of the proof is the following lemma which also acts as the “base case” of our proof.

► **Lemma 12.** *Let Δ be an upper bound on f_{rem} . Consider T values x_1, \dots, x_T chosen iid and uniformly in $(0, b)$. Consider a deterministic algorithm \mathcal{A} in CAI model that finds the minimum value y among x_1, \dots, x_T . If $T > 1000\Delta$, then with probability at least $\frac{7}{10}$ \mathcal{A} will compare y against an element x such that $x \geq b/(100\Delta)$.*

Proof. By simple scaling, we can assume $b = 1$. Let p be the probability that \mathcal{A} compares y against a value larger than $1/(100\Delta)$. Let I_{small} be the set of indices i such that $x_i < 1/(100\Delta)$. Let \mathcal{A}' be a deterministic algorithm in CAI model such that:

- \mathcal{A}' is given all the indices in I_{small} (and their corresponding values) except for the index of the minimum. We call these the known values.
- \mathcal{A}' minimizes the probability p' of comparing the y against a value larger than $1/(100\Delta)$.
- \mathcal{A}' finds the minimum value among the unknown values.

Since $p' \leq p$, it suffices to bound p' from below. We do this in the remainder of the proof.

Observe that the expected number of values x_i such that $x_i < 1/(100\Delta)$ is $T/(100\Delta)$. Thus, by Markov's inequality, $\Pr[|I_{\text{small}}| \leq T/(10\Delta)] \geq \frac{9}{10}$. Let's call the event $|I_{\text{small}}| \leq T/(10\Delta)$ the good event. For algorithm \mathcal{A}' all values smaller than $1/(100\Delta)$ except for the minimum are known. Let U be the set of indices of the unknown values. Observe that a value x_i for $i \in U$ is either the minimum or larger than $1/(100\Delta)$, and that $|U| = T - |I_{\text{small}}| + 1 > \frac{9}{10}T$ (using $\Delta \geq 1$) in the good event. Because \mathcal{A}' is a deterministic algorithm, the set U is split into set F of elements that have their first comparison against a known element, and set W of those that are first compared with another element with index in U . Because of the global bound Δ on the fragile complexity of the known elements, we know $|F| < \Delta \cdot |I_{\text{small}}| \leq \Delta T/(10\Delta) = T/10$. Combining this with the probability of the good event, by union bound, the probability of the minimum being compared with a value greater than $1/(100\Delta)$ is at least $1 - (1 - \frac{9}{10}) - (1 - \frac{8}{9}) \geq 7/10$. ◀

Based on the above lemma, our proof idea is the following. Let $G = 100\Delta$. We would like to prove that on average \mathcal{A} cannot avoid comparing the minimum to a lot of elements. In particular, we show that, with constant probability, the minimum will be compared against some value in the range $[G^{-i}, G^{-i+1}]$ for every integer i , $1 \leq i \leq \frac{\log_G n}{2}$. Our lower bound then follows by an easy application of the linearity of expectations. Proving this, however, is a little bit tricky. However, observe that Lemma 12 already proves this for $i = 1$. Next, we use the following lemma to apply Lemma 12 over all values of i , $1 \leq i \leq \frac{\log_G n}{2}$.

► **Lemma 13.** *For a value b with $0 < b < 1$, define $p_k = \binom{n}{k} b^k (1-b)^{n-k}$, for $0 \leq k \leq n$. Choosing x_1, \dots, x_n iid and uniformly in $(0, 1)$ is equivalent to the following: with probability p_k , uniformly sample a set I of k distinct indices in $\{1, \dots, n\}$ among all the subsets of size k . For each $i \in I$, pick x_i iid and uniformly in $(0, b)$. For each $i \notin I$, pick x_i iid and uniformly in $(b, 1)$.*

Proof. It is easy to see that choosing x_1, \dots, x_n iid uniformly in $(0, 1)$ is equivalent to choosing a point X uniformly at random inside an n dimensional unit cube $(0, 1)^n$. Therefore, we will prove the equivalence between (i) the distribution defined in the lemma, and (ii) choosing such point X .

Let Q be the n -dimensional unit cube. Subdivide Q into 2^n rectangular region defined by the Cartesian product of intervals $(0, b)$ and $(b, 1)$, i.e., $\{(0, b), (b, 1)\}^n$ (or alternatively, bisect Q with n hyperplanes, with the i -th hyperplane perpendicular to the i -th axis and intersecting it at coordinate equal to b).

Consider the set R_k of rectangles in $\{(0, b), (b, 1)\}^n$ with exactly k sides of length b and $n - k$ sides of length $1 - b$. Observe that for every choice of k (distinct) indices i_1, \dots, i_k out of $\{1, \dots, n\}$, there exists exactly one rectangle r in R_k such that r has side length b at dimensions i_1, \dots, i_k , and all the other sides of r has length $1 - b$. As a result, we know that the number of rectangles in R_k is $\binom{n}{k}$ and the volume of each rectangle in R_k is $b^k (1-b)^{n-k}$. Thus, if we choose a point X randomly inside Q , with probability p_k it will fall inside a rectangle r in R_k ; furthermore, conditioned on this event, the dimensions i_1, \dots, i_k where r has side length b is a uniform subset of k distinct indices from $\{1, \dots, n\}$. ◀

Remember that our goal was to prove that with constant probability, the minimum will be compared against some value in the range $[G^{-i}, G^{-i+1}]$ for every integer i , $1 \leq i \leq \frac{\log_G n}{2}$. We can pick $b = G^{-i+1}$ and apply Lemma 13. We then observe that it is very likely that the set of indices I that we are sampling in Lemma 13 will contain many indices. For every element x_i , $i \in I$, we are sampling x_i independently and uniformly in $(0, b)$ which opens the door for us to apply Lemma 12. Then we argue that Lemma 12 would imply that with constant probability the minimum will be compared against a value in the range $(b/G, b) = (G^{-i}, G^{-i+1})$. The lower bound claim of Theorem 10 then follows by invoking the linearity of expectations.

We are ready to prove that the minimum element will have $\Omega(\log_\Delta n)$ comparisons on average.

Proof of Theorem 10. First, observe that we can assume $n \geq (100,000\Delta)^2$ as otherwise we are aiming for a trivial bound of $\Omega(1)$.

We create an input set of n values x_1, \dots, x_n where each x_i is chosen iid and uniformly in $(0, 1)$. Let $G = 100\Delta$. Consider an integer i such that $1 \leq i < \frac{\log_G n}{2}$. We are going to prove that with constant probability, the minimum will be compared against a value in the range (G^{-i}, G^{-i+1}) , which, by linearity of expectation, shows the stated $\Omega(\log_\Delta n)$ lower bound for the fragile complexity of the minimum.

Consider a fixed value of i . Let S be the set of indices with values that are smaller than G^{-i+1} . Let p be the probability that \mathcal{A} compares the minimum against an x_j with $j \in S$ such that $x_j \geq G^{-i}$. To prove the theorem, it suffices to prove that p is lower bounded by a constant. Now consider an algorithm \mathcal{A}' that finds the minimum but for whom all the values other than those in S have been revealed and furthermore, assume \mathcal{A}' minimizes the probability of comparing the minimum against an element $x \geq G^{-i}$ (in other words, we pick the algorithm which minimizes this probability, among all the algorithms). Clearly, $p' \leq p$. In the rest of the proof we will give a lower bound for p' .

Observe that $|S|$ is a random variable with binomial distribution. Hence $\mathbb{E}[|S|] = nG^{-i+1} > \sqrt{n}$ where the latter follows from $i < \frac{\log_G n}{2}$. By the properties of the binomial distribution we have that $\Pr[|S| < \frac{\mathbb{E}[|S|]}{100}] < \frac{1}{10}$. Thus, with probability at least $\frac{9}{10}$, we will have the “good” event that $|S| \geq \frac{\mathbb{E}[|S|]}{100} \geq \frac{\sqrt{n}}{100}$.

In case of the good event, Lemma 13 implying that conditioned on S being the set of values smaller than G^{-i+1} , each value x_j with $j \in S$ is distributed independently and uniformly in the range $(0, G^{-i+1})$. As a result, we can now invoke Lemma 12 on the set S with $T = |S|$. Since $n \geq (100,000\Delta)^2$ we have $T = |S| \geq \frac{\sqrt{n}}{100} \geq \frac{100,000\Delta}{100}$. By Lemma 12, with probability at least $\frac{7}{10}$, the minimum will be compared against a value that is larger than G^{-i} .

Thus, by law of total probability, it follows that in case of a good event, with probability $\frac{7}{10}$ the minimum will be compared to a value in the range (G^{-i}, G^{-i+1}) . However, as the good event happens with probability $\frac{9}{10}$, it follows that with probability at least $1 - (1 - \frac{7}{10}) - (1 - \frac{9}{10}) = \frac{6}{10}$, the minimum will be compared against a value in the range (G^{-i}, G^{-i+1}) . ◀

2.3.2 Lower bound for the fragile complexity of the minimum whp.

With Theorem 8 in Subsection 2.2, we show in particular that `SAMPLEMINIMUM` guarantees that the fragile complexity of the minimum is at most $\mathcal{O}(\log \log n)$ with probability at least $1 - 1/n^c$ for any $c > 1$. (By setting $t = 2 \log \log n$).

Here we show that this is optimal up to constant factors in the fragile complexity.

► **Theorem 14.** *For any constant $\varepsilon > 0$, there exists a value of n_0 such that the following holds for any randomized algorithm \mathcal{A} and for any $n > n_0$: there exists an input of size n such that with probability at least $n^{-\varepsilon}$, \mathcal{A} performs $\geq \frac{1}{2} \log \log n$ comparisons with the minimum.*

Proof. We use (again) Yao’s principle and consider a fixed deterministic algorithm \mathcal{A} working on the uniform input distribution, i.e., all input permutations have probability $1/n!$. Let $f = \frac{1}{2} \log \log n$ be the upper bound on the fragile complexity of the minimum. Let $k = 2^f = \sqrt{\log n}$ and let S be the set of the k smallest input values. Let π be a uniform permutation (the input) and $\pi(S)$ be the permutation of the elements of S in π . Observe that $\pi(S)$ is a uniform permutation of the elements of S . We reveal the elements not in S to \mathcal{A} . So, \mathcal{A} only needs to find the minimum in $\pi(S)$. By Theorem 2 there is at least one “bad” permutation of S which forces algorithm \mathcal{A} to do $\log k = f$ comparisons on the smallest element. Observe $\log k! < \log k^k = k \log k = \sqrt{\log n} \frac{1}{2} \log \log n$. Observe that there exists a value of n_0 such that for $n > n_0$ the right hand side is upper bounded by $\varepsilon \log n$, so $k! \leq n^\varepsilon$, for $n > n_0$. Hence, the probability of a “bad” permutation is at least $1/k! > n^{-\varepsilon}$. ◀

3 Selection and median

The (n, t) -selection problem asks to find the t -th smallest element among n elements of the input. The simplest solution to the (n, t) -selection problem is to sort the input. Therefore, it can be solved in $\mathcal{O}(\log n)$ fragile complexity and $\mathcal{O}(n \log n)$ work by using the AKS sorting network [2]. For comparator networks, both of these bounds are optimal: the former is shown by Theorem 2 (and in fact it applies also to any algorithm) and the latter is shown in the full version of this paper [1].

In contrast, in this section we show that comparison-based algorithms can do better: we can solve SELECTION deterministically in $\Theta(n)$ work and $\Theta(\log n)$ fragile complexity, thus, showing a separation between the two models. However, to do that, we resort to constructions that are based on expander graphs. Avoiding usage of the expander graphs or finding a simpler optimal deterministic solution is an interesting open problem (see Section 6). Moreover, in Subsection 3.2 we show that we can do even better by using randomization.

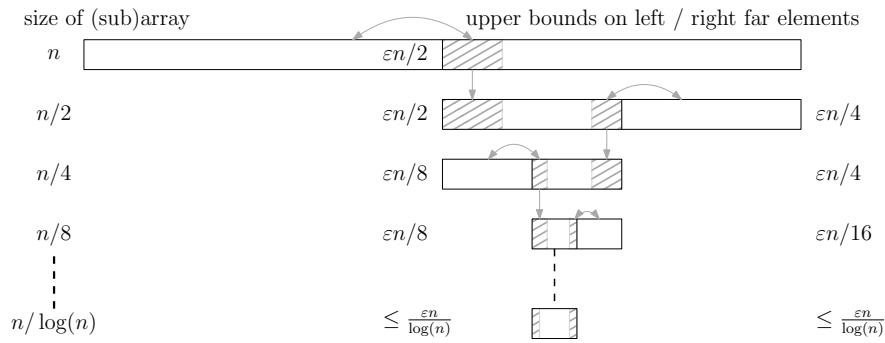
3.1 Deterministic selection

► **Theorem 15.** *There is a deterministic algorithm for SELECTION which performs $\mathcal{O}(n)$ work and has $\mathcal{O}(\log n)$ fragile complexity.*

Proof sketch. It suffices to just find the median since by simple padding we can generalize the solution for the (n, t) -selection problem.

We use ε -halvers that are the central building blocks of the AKS sorting network. An ε -halver approximately performs a partitioning of an array of size n into the smallest half and the largest half of the elements. More precisely, for any $m \leq n/2$, at most εn of the m smallest elements will end up in the right half of the array, and at most εn of the m largest elements will end up in the left half of the array. Using expander graphs, a comparator network implementing an ε -halver of constant depth can be built [2, 4]. We use the corresponding comparison-based algorithm of constant fragile complexity.

The idea is to use ε -halvers to find roughly $\frac{n}{\log n}$ elements with rank between $(1 + \alpha) \frac{n}{2}$ and $(2 - \alpha) \frac{n}{2}$ and also roughly $\frac{n}{\log n}$ elements with rank between $\alpha \frac{n}{2}$ and $(1 - \alpha) \frac{n}{2}$, for some constant $0 < \alpha < 1$. This is done by repeatedly using ε -halvers but alternating between



■ **Figure 1** Illustration of the alternating division process using ε -halvers.

selecting the left half and the right half (Figure 1). Using these, we filter the remaining elements and discard a constant fraction of them. Then we recurse on the remaining elements. The details are a bit involved, as we have to guarantee that no element accumulates too many comparisons throughout the recursions. We have to do some bookkeeping as well as some additional ideas to provide this guarantee. Details can be found in the full version of the paper [1]. ◀

► **Corollary 16.** *There is a deterministic algorithm for partition which performs $\mathcal{O}(n)$ work and has $\mathcal{O}(\log n)$ fragile complexity.*

Proof. At the end of the SELECTION algorithm, the set of elements smaller (larger) than the median is the union of the respective filtered sets (sets \mathcal{L} and \mathcal{R} in the proof in the full version of the paper [1]) and the first (last) half of the sorted set in the base case of the recursion. Again, simple padding generalizes this to (n, t) -partition for arbitrary $t \neq \frac{n}{2}$. ◀

3.2 Randomized selection

In the full paper [1], we present the details of an expected work-optimal selection algorithm with a trade-off between the expected fragile complexity $f_{\text{med}}(n)$ of the selected element and the maximum expected fragile complexity $f_{\text{rem}}(n)$ of the remaining elements. In particular, we obtain the following combinations:

► **Theorem 17.** *Randomized selection is possible in expected linear work, while achieving expected fragile complexity of the median $\mathbb{E}[f_{\text{med}}(n)] = \mathcal{O}(\log \log n)$ and of the remaining elements $\mathbb{E}[f_{\text{rem}}(n)] = \mathcal{O}(\sqrt{n})$, or $\mathbb{E}[f_{\text{med}}(n)] = \mathcal{O}\left(\frac{\log n}{\log \log n}\right)$ and $\mathbb{E}[f_{\text{rem}}(n)] = \mathcal{O}(\log^2 n)$.*

4 Sorting

Recall from Section 1 that the few existing sorting networks with depth $\mathcal{O}(\log n)$ are all based on expanders, while a number of $\mathcal{O}(\log^2 n)$ depth networks have been developed based on binary merging. Here, we study the power of the mergesort paradigm with respect to fragile complexity. We first prove that any sorting algorithm based on binary merging must have a worst-case fragile complexity of $\Omega(\log^2 n)$. This provides an explanation why all existing sorting networks based on merging have a depth no better than this. We also prove that the standard mergesort algorithm on random input has fragile complexity $\mathcal{O}(\log n)$ with high probability, thereby showing a separation between the deterministic and the randomized situation for binary mergesorts. Finally, we demonstrate that the standard mergesort

algorithm has a worst-case fragile complexity of $\Theta(n)$, but that this can be improved to $\mathcal{O}(\log^2 n)$ by changing the merging algorithm to use exponential search. The omitted proofs can be found in the full paper [1].

► **Lemma 18.** *Merging of two sorted sequences A and B has fragile complexity at least $\lfloor \log_2 |A| \rfloor + 1$.*

► **Theorem 19.** *Any binary mergesort has fragile complexity $\Omega(\log^2 n)$.*

Proof. The adversary is the same as in the proof of Lemma 21, except that as scapegoat element for a merge of A and B it always chooses the scapegoat from the *larger* of A and B . We claim that for this adversary, there is a constant $c > 0$ such that for any node v in the mergetree, its scapegoat element has participated in at least $c \log^2 n$ comparisons in the subtree of v , where n is the number of elements merged by v . This implies the theorem.

We prove the claim by induction on n . The base case is $n = \mathcal{O}(1)$, where the claim is true for small enough c , as the scapegoat by Lemma 18 will have participated in at least one comparison. For the induction step, assume v merges two sequences of sizes n_1 and n_2 , with $n_1 \geq n_2$. By the base case, we can assume $n_1 \geq 3$. Using Lemma 18, we would like to prove for the induction step

$$c \log^2 n_1 + \lfloor \log n_2 \rfloor + 1 \geq c \log^2(n_1 + n_2). \quad (1)$$

This will follow if we can prove

$$\log^2 n_1 + \frac{\log n_2}{c} \geq \log^2(n_1 + n_2). \quad (2)$$

The function $f(x) = \log^2 x$ has first derivative $2(\log x)/x$ and second derivative $2(1 - \log x)/x^2$, which is negative for $x > e = 2.71\dots$. Hence, $f(x)$ is concave for $x > e$, which means that first order Taylor expansion (alias the tangent) lies above f , i.e., $f(x_0) + f'(x_0)(x - x_0) \geq f(x)$ for $x_0, x > e$. Using $x_0 = n_1$ and $x = n_1 + n_2$ and substituting the first order Taylor expansion into the right side of (2), we see that (2) will follow if we can prove

$$\frac{\log n_2}{c} \geq 2 \frac{\log n_1}{n_1} n_2,$$

which is equivalent to

$$\frac{\log n_2}{n_2} \geq 2c \frac{\log n_1}{n_1}. \quad (3)$$

Since $n_1 \geq n_2$ and $(\log x)/x$ is decreasing for $x \geq e$, we see that (3) is true for $n_2 \geq 3$ and c small enough. Since $\log(3)/3 = 0.366\dots$ and $\log 2/2 = 0.346\dots$, it is also true for $n_2 = 2$ and c small enough. For the final case of $n_2 = 1$, the original inequality (1) reduces to

$$\log^2 n_1 + \frac{1}{c} \geq \log^2(n_1 + 1). \quad (4)$$

Here we can again use concavity and first order Taylor approximation with $x_0 = n_1$ and $x = n_1 + 1$ to argue that (4) follows from

$$\frac{1}{c} \geq 2 \frac{\log n_1}{n_1}.$$

which is true for c small enough, as $n_1 \geq 3$ and $(\log x)/x$ is decreasing for $x \geq e$. ◀

► **Theorem 20.** *Standard MERGESORT with linear merging has a worst-case fragile complexity of $\Theta(n)$.*

► **Lemma 21.** *Standard MERGESORT has fragile complexity $\Omega(\log^2 n)$.*

Proof. In MERGESORT, when merging two sorted sequences A and B , no comparisons between elements of A and B have taken place before the merge. Also, the sorted order of $A \cup B$ has to be decided by the algorithm after the merge. We can therefore run the adversary argument from the proof of Lemma 18 in all nodes of the mergetree of MERGESORT. If the adversary reuses scapegoat elements in a bottom-up fashion – that is, as scapegoat for a merge of A and B chooses one of the two scapegoats from the two merges producing A and B – then the scapegoat at the root of the mergetree has participated in

$$\Omega\left(\sum_{i=0}^{\log n} \log 2^i\right) = \Omega\left(\sum_{i=0}^{\log n} i\right) = \Omega(\log^2 n)$$

comparisons, by Lemma 18 and the fact that a node at height i in the mergetree of standard MERGESORT operates on sequences of length $\Theta(2^i)$. ◀

► **Observation 1.** *Consider two sorted sequences $A = (a_1, \dots, a_n)$ and $B = (b_1, \dots, b_n)$. In linear merging, the fragile complexity of element a_i is at most $\ell + 1$ where ℓ is the largest number of elements from B that are placed directly in front of a_i (i.e. $b_j < \dots < b_{j+\ell-1} < a_i$).*

► **Lemma 22.** *Let $X = \{x_1, \dots, x_{2k}\}$ be a finite set of distinct elements, and consider a random bipartition $X_L, X_R \subset X$ with $|X_L| = |X_R| = k$ and $X_L \cap X_R = \emptyset$, such that $\Pr[x_i \in X_L] = 1/2$. Consider an arbitrary ordered set $Y = \{y_1, \dots, y_m\} \subset X$ with $m \leq k$. Then $\Pr[Y \subseteq X_L \vee Y \subseteq X_R] < 2^{1-m}$.*

Proof.

$$\Pr[Y \subseteq X_L \vee Y \subseteq X_R] = 2 \prod_{i=1}^m \Pr[y_i \in X_L \mid y_1, \dots, y_{i-1} \in X_L] = 2 \frac{(2k)^{-m} k!}{(k-m)!} \leq 2 \cdot 2^{-m}.$$

◀

► **Theorem 23.** *Standard MERGESORT with linear merging on a randomized input permutation has a fragile complexity of $\mathcal{O}(\log n)$ with high probability.*

Proof. Let $Y = (y_1, \dots, y_n)$ be the input-sequence, π^{-1} be the permutation that sorts Y and $X = (x_1, \dots, x_n)$ with $x_i = y_{\pi^{-1}(i)}$ be the sorted sequence. Wlog we assume that all elements are unique², that any input permutation π is equally likely³, and that n is a power of two.

Merging in one layer. Consider any merging-step in the mergetree. Since both input sequences are sorted, the only information still observable from the initial permutation is the bi-partitioning of elements into the two subproblems. Given π , we can uniquely retrace the mergetree (and vice-versa): we identify each node in the recursion tree with the set of elements it considers. Then, any node with elements $X_P = \{y_\ell, \dots, y_{\ell+2k-1}\}$ has children

$$\begin{aligned} X_L &= \{x_{\pi(i)} \mid \ell \leq \pi(i) \leq \ell + k - 1\} = \{y_\ell, \dots, y_{\ell+k-1}\}, \\ X_R &= \{x_{\pi(i)} \mid \ell + k \leq \pi(i) \leq \ell + 2k - 1\} = \{y_{\ell+k}, \dots, y_{\ell+2k-1}\}. \end{aligned}$$

² If this is not the case, use input sequence $Y' = ((y_1, 1), \dots, (y_n, n))$ and lexicographical compares.

³ If not shuffle it before sorting in linear time and no fragile comparisons.

Hence, locally our input permutation corresponds to an stochastic experiment in which we randomly draw exactly half of the parent's elements for the left child, while the remainder goes to right.

This is exactly the situation in Lemma 22. Let N_i be a random variable denoting the number of comparisons of element y_i in the merging step. Then, from Observation 1 and Lemma 22 it follows that $\Pr[N_i = m+1] \leq 2^{-m}$. Therefore N_i is stochastically dominated by $N_i \preceq 1+Y_i$ where Y_i is a geometric random variable with success probability $p = 1/2$.

Merging in all layers. Let $N_{j,i}$ be the number of times element y_i is compared in the j -th recursion layer and define $Y_{j,i}$ analogously. Due to the recursive partitioning argument, $N_{j,i}$ and $Y_{j,i}$ are iid in j . Let N_i^T be the total number of comparisons of element i , i.e. $N_i^T \preceq \log_2 n + \sum_{j=1}^{\log_2 n} Y_{j,i}$. Then a tail bound on the sum of geometric variables (Theorem 2.1 in [13]) yields:

$$\Pr \left[\sum_{j=1}^{\log_2 n} Y_{j,i} \geq \lambda \mathbb{E} \left[\sum_{j=1}^{\log_2 n} Y_{j,i} \right] = 2\lambda \log_2 n \right] \stackrel{[13]}{\leq} \exp \left(-\frac{1}{2} \frac{2 \ln n}{\ln 2} [\lambda - 1 - \log \lambda] \right) = n^{-2},$$

where we set $\lambda \approx 3.69$ in the last step solving $\lambda - \log \lambda = 2 \log 2$. Thus, we bound the probability $\Pr[N_i^T \geq (1+2\lambda) \log_2 n] \leq n^{-2}$.

Fragile complexity. It remains to show that with high probability no element exceeds the claimed fragile complexity. We use a union bound on N_i^T for all i :

$$\Pr \left[\max_i \{N_i^T\} = \omega(\log n) \right] \leq n \Pr[N_i^T = \omega(\log n)] \leq 1/n. \quad \blacktriangleleft$$

► **Theorem 24.** *Exponential merging of two sequences $A = (a_1, \dots, a_n)$ and $B = (b_1, \dots, b_n)$ has a worst-case fragile complexity of $\mathcal{O}(\log n)$.*

► **Corollary 25.** *Applying Theorem 24 to standard MERGESORT with exponential merging yields a fragile complexity of $\mathcal{O}(\log^2 n)$ in the worst-case.*

5 Constructing binary heaps

► **Observation 2.** *The fragile complexity of the standard binary heap construction algorithm of Floyd [11] is $\mathcal{O}(\log n)$.*

The above observation is easy to verify (shown in the full paper [1]). We note that this fragile complexity is optimal by Theorem 2, since HEAP CONSTRUCTION is stronger than MINIMUM. Brodal and Pinotti [7] showed how to construct a binary heap using a comparator network in $\Theta(n \log \log n)$ size and $\mathcal{O}(\log n)$ depth. They also proved a matching lower bound on the size of the comparator network for this problem. This, together with Observation 2 and the fact that Floyd's algorithm has work $\mathcal{O}(n)$, gives a separation between work of fragility-optimal comparison-based algorithms and size of depth-optimal comparator networks for HEAP CONSTRUCTION.

6 Conclusions

In this paper we introduced the notion of fragile complexity of comparison-based algorithms and we argued that the concept is well-motivated because of connections both to real world situations (e.g., sporting events), as well as other fundamental theoretical concepts (e.g., sorting networks). We studied the fragile complexity of some of the fundamental problems and revealed interesting behavior such as the large gap between the performance of deterministic and randomized algorithms for finding the minimum. We believe there are still plenty of interesting and fundamental problems left open. Below, we briefly review a few of them.


- The area of comparison-based algorithms is much larger than what we have studied. In particular, it would be interesting to study “geometric orthogonal problems” such as finding the maxima of a set of points, detecting intersections between vertical and horizontal line segments, kd -trees, axis-aligned point location and so on. All of these problems can be solved using algorithms that simply compare the coordinates of points.
- Is it possible to avoid using expander graphs to obtain simple deterministic algorithms to find the median or to sort?
- Is it possible to obtain a randomized algorithm that finds the median where the median suffers $O(1)$ comparisons on average? Or alternatively, is it possible to prove a lower bound? If one cannot show a $\omega(1)$ lower bound for the fragile complexity of the median, can we show it for some other similar problem?

References

- 1 P. Afshani, R. Fagerberg, D. Hammer, R. Jacob, I. Kostitsyna, U. Meyer, M. Penschuck, and N. Sitchinava. Fragile Complexity of Comparison-Based Algorithms. *CoRR*, abs/1901.02857, 2019. [arXiv:1901.02857](https://arxiv.org/abs/1901.02857).
- 2 M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ Sorting Network. In *Proceedings of the 15th Symposium on Theory of Computation*, STOC '83, pages 1–9. ACM, 1983. doi:10.1145/800061.808726.
- 3 M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3(1):1–19, March 1983. doi:10.1007/BF02579338.
- 4 M. Ajtai, J. Komlós, and E. Szemerédi. Halvers and Expanders. In IEEE, editor, *FOCS'92*, pages 686–692, Pittsburgh, PN, October 1992. IEEE Computer Society Press.
- 5 V. E. Alekseev. Sorting Algorithms with minimum memory. *Kibernetika*, 5(5):99–103, 1969.
- 6 K. E. Batcher. Sorting Networks and their Applications. *Proceedings of AFIPS Spring Joint Computer Conference*, pages 307–314, 1968.
- 7 G. Stølting Brodal and M. C. Pinotti. Comparator Networks for Binary Heap Construction. In *Proc. 6th Scandinavian Workshop on Algorithm Theory*, volume 1432 of *LNCS*, pages 158–168. Springer Verlag, Berlin, 1998. doi:10.1007/BFb0054364.
- 8 V. Chvátal. Lecture notes on the new AKS sorting network. Technical Report DCS-TR-294, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1992, October.
- 9 R. Cole. Parallel Merge Sort. *SIAM Journal on Computing*, 17(4):770–785, 1988.
- 10 M. Dowd, Y. Perl, L. Rudolph, and M. Saks. The periodic balanced sorting network. *J. ACM*, 36(4):738–757, 1989, October.
- 11 R. W. Floyd. Algorithm 245: Treesort. *Commun. ACM*, 7(12):701, December 1964. doi:10.1145/355588.365103.
- 12 M. T. Goodrich. Zig-zag sort: a simple deterministic data-oblivious sorting algorithm running in $O(n \log n)$ time. In David B. Shmoys, editor, *STOC'14*, pages 684–693. ACM, 2014. URL: <http://dl.acm.org/citation.cfm?id=2591796>.
- 13 S. Janson. Tail bounds for sums of geometric and exponential variables. *Statistics & Probability Letters*, 135:1–6, 2018. doi:10.1016/j.spl.2017.11.017.
- 14 S. Jimbo and A. Maruoka. A Method of Constructing Selection Networks with $O(\log n)$ Depth. *SIAM Journal on Computing*, 25(4):709–739, 1996.
- 15 I. Parberry. The pairwise sorting network. *Parallel Processing Letters*, 2(2-3):205–211, 1992.
- 16 B. Parker and I. Parberry. Constructing sorting networks from k -sorters. *Information Processing Letters*, 33(3):157–162, 30 nov 1989.
- 17 M. S. Paterson. Improved sorting networks with $O(\log N)$ depth. *Algorithmica*, 5(1):75–92, 1990.
- 18 N. Pippenger. Selection Networks. *SIAM Journal on Computing*, 20(5):878–887, 1991.

- 19 V. R. Pratt. *Shellsort and Sorting Networks*. Outstanding Dissertations in the Computer Sciences. Garland Publishing, New York, 1972.
- 20 J. I. Seiferas. Sorting Networks of Logarithmic Depth, Further Simplified. *Algorithmica*, 53(3):374–384, 2009.
- 21 S. Hoory, N. Linial, and A. Wigderson. Expander Graphs and Their Applications. *BAMS: Bulletin of the American Mathematical Society*, 43:439–561, 2006.
- 22 S. P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012.
- 23 A. Yao and F. F. Yao. Lower Bounds on Merging Networks. *J. ACM*, 23(3):566–571, 1976.

Universal Reconfiguration of Facet-Connected Modular Robots by Pivots: The $O(1)$ Musketeers

Hugo A. Akitaya 

Tufts University, Medford,
MA, USA
hugo.alves_akitaya@tufts.edu

Mirela Damian

Villanova University,
PA, USA
mirela.damian@villanova.edu

Vida Dujmović

University of Ottawa, Canada
vida.dujmovic@uottawa.ca

Matias Korman

Tufts University, Medford, MA, USA
Matias.Korman@tufts.edu

Irene Parada 

Graz University of Technology, Austria
iparada@ist.tugraz.at

Vera Sacristán 

Universitat Politècnica de Catalunya,
Barcelona, Spain
vera.sacristan@upc.edu

Esther M. Arkin 

State University of New York at Stony Brook,
NY, USA
esther.arkin@stonybrook.edu

Erik D. Demaine 

Massachusetts Institute of Technology,
Cambridge, MA, USA
edemaine@mit.edu

Robin Flatland

Siena College, Loudonville, NY, USA
flatland@siena.edu

Belen Palop

Universidad de Valladolid, Spain
belen.palop@uva.es

André van Renssen 

The University of Sydney, Australia
andre.vanrenssen@sydney.edu.au

Abstract

We present the first universal reconfiguration algorithm for transforming a modular robot between any two facet-connected square-grid configurations using pivot moves. More precisely, we show that five extra “helper” modules (“musketeers”) suffice to reconfigure the remaining n modules between any two given configurations. Our algorithm uses $O(n^2)$ pivot moves, which is worst-case optimal. Previous reconfiguration algorithms either require less restrictive “sliding” moves, do not preserve facet-connectivity, or for the setting we consider, could only handle a small subset of configurations defined by a local forbidden pattern. Configurations with the forbidden pattern do have disconnected reconfiguration graphs (discrete configuration spaces), and indeed we show that they can have an exponential number of connected components. But forbidding the local pattern throughout the configuration is far from necessary, as we show that just a constant number of added modules (placed to be freely reconfigurable) suffice for universal reconfigurability. We also classify three different models of natural pivot moves that preserve facet-connectivity, and show separations between these models.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Reconfiguration, geometric algorithm, pivoting squares, modular robots

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.3

Related Version A full version of the paper is available at <https://arxiv.org/abs/1908.07880>.

Funding *Hugo A. Akitaya*: Supported by NSF CCF-1422311 and CCF-1423615.

Esther M. Arkin: Partially funded by NSF (CCF-1526406).

Erik D. Demaine: Supported in part by NSF ODISSEI grant EFRI-1240383 and NSF Expedition grant CCF-1138967.

Belen Palop: Partially supported by MTM2015-63791-R (MINECO/FEDER).

Irene Parada: Supported by the Austrian Science Fund (FWF): W1230.



© Hugo A. Akitaya, Esther M. Arkin, Mirela Damian, Erik D. Demaine, Vida Dujmović, Robin Flatland, Matias Korman, Belen Palop, Irene Parada, André van Renssen, and Vera Sacristán; licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 3; pp. 3:1–3:14



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

André van Renssen: Supported by JST ERATO Grant Number JPMJER1201, Japan.

Vera Sacristán: Partially supported by MTM2015-63791-R (MINECO/FEDER) and Gen. Cat. DGR 2017SGR1640.

Acknowledgements This research started at the 32nd Bellairs Winter Workshop on Computational Geometry in 2017. We want to thank all participants for the fruitful discussions on the topic. We would also like to thank the reviewers for their insightful and valuable comments.

1 Introduction

Shape shifting is a powerful idea in science fiction: T-1000 robots (from *Terminator 2: Judgement Day*), Changelings (from *Star Trek: Deep Space 9*), Symbiotes (from *Venom*), Mystique (from *X-Men*), and Metamorphagi (from *Harry Potter*) all have the ability to transform their shape nearly arbitrarily. How can we make shape shifting into science?

Modular robots [5, 19, 22] are perhaps the best answer to this question. The idea to build a single “robot” out of many small units called *modules*, each of which can attach and detach from each other, move relative to each other, communicate with each other, and compute. Modular robots offer extreme adaptability to changing environment or user needs, in particular by reconfiguring the modules into exponentially many effective shapes of the overall robot. Modularity also offers a practical future for manufacturing (identical modules can be mass-produced, making them relatively cheap) makes robots easy to repair by just replacing the broken modules, and makes it possible to re-use components from one robot/task to another.

For computational geometry, modular robots offer exciting challenges: what shapes can a modular robot self-reconfigure into, and what are good algorithms for reconfiguration? According to [19], the main difficulties in self-reconfiguration are the physical motion constraints of the modules themselves, connectivity requirements for the robot to hold together, collisions between moving and/or static modules, and “deadlocks” where no module can move or some module gets “trapped” within the configuration.

The wide diversity of mechatronic solutions to modular robots can be characterized from a geometric viewpoint by three key properties: (1) the lattice, (2) connectivity requirement, and (2) allowed moves.

Lattice. Most modular robots follow a space-filling lattice structure (e.g., squares or hexagons in 2D, or cubes in 3D), to simplify both reconfiguration and the characterization of possible shapes. Pure lattice modular robots [13, 6, 10, 17, 2, 20] have one robot per lattice element and always remain on the lattice, while hybrid modular robots [14, 18, 16, 23] also allow units move out of the lattice. We focus here on the well-studied square lattice, though we suspect our results can be generalized to cube lattices.

Connectivity requirement. A modular robot generally needs to be connected at all times while reconfiguring, so that the modules do not fall apart. The most common and practical constraint is that the modules are always *facet-connected*, meaning a connected *facet-adjacency graph* where vertices represent modules and edges represent adjacencies by shared facets (edges in 2D). The exception is that the moving module is excluded from this graph during each move, meaning that other modules must be facet-connected while the moving module may briefly disconnect during the move. A weaker connectivity constraint, considered in some theoretical research [4, Ch. 4], is that the robot is connected via shared vertices. In such case, reconfiguration is always possible. We focus here on the more challenging facet-connectivity constraint.

Allowed moves. One of the most popular models is *sliding squares/cubes* [8, 7, 1], illustrated in Figure 1 left. In this case, modules live in a square or cube lattice, move by sliding relative to each other, and require facet-connectivity. For this model, universal reconfiguration is possible between any two facet-connected configurations, in any dimension [7, 1].



■ **Figure 1** Two ways a module a starting above module s can move to the adjacent lattice position, above module s' . Left: sliding. Right: pivoting. Pivoting requires more free space to execute.

We focus here on a more challenging model, *pivoting squares/cubes* [21, 20, 4], illustrated in Figure 1 right. In this case, modules live in a square or cube lattice, move by rotating relative to each other, and require facet-connectivity. The key difference is that a module needs two additional squares/cubes of empty space in order to pivot, whereas a slide just needs the destination square/cube to be empty. Unfortunately, some configurations are *rigid* in this model, meaning that no module can move without disconnecting the robot.

Rigid configurations appear also in the sliding square model when the sliding capability is restricted to turning corners [12]. However, in this model the existence of free space around the modules does not guarantee reconfigurability, while in the pivoting squares model it does, as we will discuss.

As a consequence, all known reconfiguration algorithms for pivoting squares/cubes are somehow partial. One algorithm follows some heuristics without a termination guarantee [3] (see also [11] for heuristics for hexagons). A recent algorithm guarantees reconfiguration by forbidding one or more local patterns in both the start and goal configurations [20], essentially preventing narrow holes in the shape. (A similar result was obtained for hexagons [15].) These assumptions severely restrict the possible shapes that can be reconfigured, to a $o(1)$ fraction. The absence of such local patterns though is far from being necessary for reconfigurability. In 3D, some further strong conditions are added, such as that every hole must be orthogonally convex [20].

Our results. Our main result is that *universal* reconfiguration is possible if we allow the addition of a constant number of (five) extra “helper” modules, which we call *musketeer modules*.¹ The key is that these musketeer modules are not considered part of the initial or target shape, and thus we are free to place them where we like (in particular, along the external boundary of the robot). Surprisingly, this small amount of additional freedom is enough to achieve universal reconfiguration. In fact, we prove in Section 4 that five musketeer modules are both sufficient and sometimes necessary to solve any reconfiguration under our strategy. Our algorithm is based on the old idea of following the right-hand rule to escape a maze [9]. The number of pivots it makes is $O(n^2)$, which is optimal in the worst case by an earth-moving lower bound: each robot may need to move a distance of $\Theta(n)$.

This result can be seen as proving connectivity of the *reconfiguration graph* $\mathcal{G}_{n,k}$, where vertices represent facet-connected configurations of n modules and edges represent valid pivot moves, with the addition of $k \geq 5$ musketeer modules. With $k = 0$ musketeers, $\mathcal{G}_{n,k}$ is known to be disconnected. Surprisingly, there have been no (successful) attempts to understand

¹ *The Three Musketeers* is a story about four musketeers. This paper is a story about five musketeers.

the structure of this reconfiguration graph. In Section 3, we analyze the structure of this reconfiguration graph. Specifically, we prove that $\mathcal{G}_{n,0}$ can have an exponential number of connected components of exponential size, and in some models, can have an exponential number of singleton connected components (rigid configurations); while in other models, the reconfiguration graph cannot have any singleton connected components.

The other main contribution of this paper is to precisely define a variety of natural models for pivot moves. Pivoting is naturally defined as the rotation of one module about one of its vertices that is shared with a (static) module. But there are some subtleties in this definition depending on exactly which modules must be facet-connected at what times. (Obviously, for example, the moving module is not facet-connected to the others during the move.) In Section 2, we define three nested models, each at least as powerful as the previous, and in Section 3, we prove strict separations between these models. Our analysis of connected components in the reconfiguration space (in Section 3) also consider the effects of these different models. We conclude with open problems in Section 5.

2 Models and Definitions

2.1 Pivot Moves

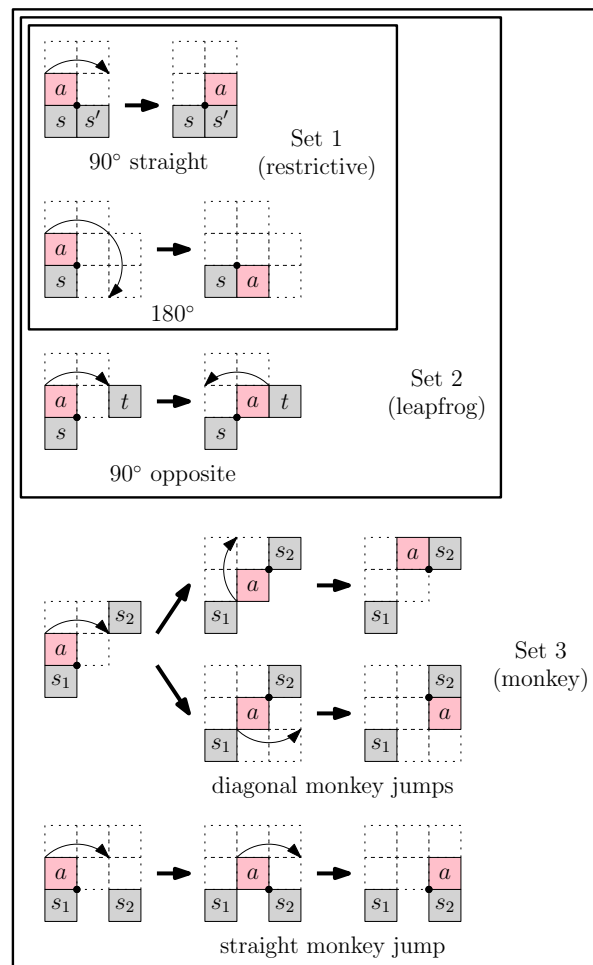
In a square grid, the fact that two squares may share a vertex without actually sharing an edge opens a wider range of possibilities for the pivoting move. Refer to Figure 2. The most restrictive set of moves (Set 1 in Figure 2) requires module a to be facet-adjacent to module s and to rotate about one of the two vertices of the edge they share. Such move can be a 90° or a 180° rotation, depending on whether or not s has a neighboring module s' adjacent to it through the other edge of s incident to the rotation center, and of course, requires the goal grid position to be empty and some intermediate positions to be (at least partially) clear. These cells are depicted in white in Figure 2.

The authors of [20] propose an expanded set of moves (Set 2 in Figure 2) that allows module a to rotate 90° about module s even when s' is not present, as long as module a is again facet-adjacent to another module t at the end of the move. Since their reconfiguration algorithm relies on reversible moves, this implies allowing also the reverse move: module a can rotate 90° about a vertex of another module s incident to a , without requiring s to be facet-adjacent to a , as long as a is facet-adjacent to some module before performing the move and after performing the move. We call this enlarged set the *leapfrog* set of moves.

If the previous move is allowed (i.e., if it is feasible for a given modular robot prototype), it seems natural to allow concatenating more than one of such moves, i.e., to allow concatenating consecutive rotations about vertices incident to the pivoting module. It is easy to prove that such concatenation cannot involve more than two pivots before the moving module becomes facet-adjacent to another module. Indeed, if a module a is facet-adjacent to a module s_1 , after at most two such moves it necessarily becomes adjacent to a module s_2 (Set 3 in Figure 2). We call this complete set the *monkey* set of moves.

2.2 Reconfiguration Problem

Consider a configuration C of n robot modules in a given grid. The *facet-adjacency graph* of C has a node for each module, and an edge between a pair of nodes if the corresponding modules are facet-adjacent. Throughout this paper we will often refer to the facet-adjacency graph simply as the *adjacency graph*. We will say that a configuration C is *facet-connected* if the facet-adjacency graph of C is connected.



■ **Figure 2** The possible sets of moves for a pivoting module a about a module s , in a square grid.

Applying a pivot move from one of the three sets of moves described in the previous section to a facet-connected configuration C , means applying one of the moves to a module in C , in such a way that the configuration (without the pivoting module) stays facet-connected before, after, and during the move, and the pivoting module does not collide with any other module. Note that this implies that even deleting the moving module the configuration is still facet-connected. Reconfiguring C consists of applying a concatenation of such moves.

The (universal) reconfiguration problem asks whether it is possible to reconfigure any facet-connected configuration of n modules in a given grid into any other configuration with the same number of modules.

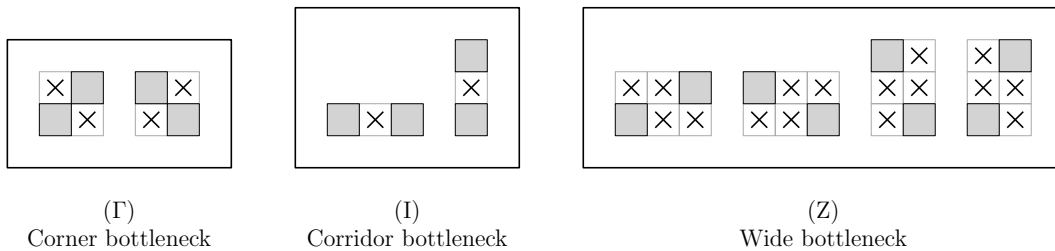
For any positive integer n , the *reconfiguration graph* \mathcal{G}_n has a node for each facet-connected configuration with n modules, and an edge between two nodes if the corresponding configurations can be reconfigured into each other through a single pivoting move. We call *rigid* any configuration in which no module can move, i.e., any configuration that is an isolated node of \mathcal{G}_n , forming a connected component that is a singleton. We call *locked* any configuration that cannot be reconfigured into a straight strip of modules, i.e., any configuration belonging to a connected component of \mathcal{G}_n that does not contain a strip.



■ **Figure 3** Left: a rigid configuration of edge-connected pivoting squares. Right: A configuration that can be reconfigured into a strip, in spite of containing instances of the three forbidden patterns.

3 Reconfiguration Graph

Figure 3 (left) shows an example of a configuration that is rigid under the largest possible set of pivoting moves (set 3 in Figure 2). In [20] it is proved that reconfiguration for set 2 of pivoting moves (leapfrog moves) is possible between two facet-connected configurations of the same number of squares, provided that they are both *admissible* shapes. Admissibility is defined in terms of forbidden patterns: a facet-connected configuration of squares is admissible if it does not contain any of the patterns depicted in Figure 4. However, this local separation condition is certainly not necessary, as proves the example in Figure 3 (right).



■ **Figure 4** The three forbidden patterns for facet-connected pivoting squares; solid squares represent modules, and \times -ed squares represent empty spaces.

These results raise several natural questions for facet-connected pivoting squares: Are the three sets of moves equivalent? In particular, is reconfigurability between admissible shapes also guaranteed when using the most restrictive set of pivoting moves? This latter question has been answered positively by the results from [20]. Although not explicitly stated, the reconfiguration algorithm from [20] uses only restrictive moves.

Several other interesting questions are open. Can the admissible condition be relaxed when using the largest set of pivoting moves? Do there exist rigid configurations that contain only one type of pattern? If so, are they rigid with respect to all three sets of pivoting moves? What can we say about the reconfiguration graph \mathcal{G}_n for the different sets of pivoting moves? We try to answer these questions in the remaining of this section. Due to space constraints, the proofs of the propositions in this section are omitted.

We start by showing that the three sets of moves for pivoting squares are not equivalent, as they produce three different reconfiguration graphs.

► **Proposition 1.** *The monkey set of moves for pivoting squares (set 3) is stronger than the leapfrog set (set 2), and the leapfrog set is stronger than the restrictive set (set 1). That is, the resulting reconfiguration graph \mathcal{G}_n has strictly fewer connected components for set 3 than for set 2, and fewer connected components for set 2 than for set 1.*

Let us now discuss the differences between the three forbidden patterns. From a purely geometric viewpoint, pattern Γ produces a (corner) bottleneck along the boundary of a configuration that is narrower than the one produced by pattern I (corridor bottleneck). This one is in turn narrower than the one produced by pattern Z (wide bottleneck). The next propositions show how the presence or the absence of each of such patterns influences reconfiguration under each of the 3 sets of pivoting moves.

Pattern Γ : Corner Bottleneck

We start by showing that pattern Γ alone suffices to make a configuration rigid, regardless of the set of pivoting moves used (restrictive, leapfrog, or monkey).

► **Proposition 2.** *Let \mathcal{G}_n be the reconfiguration graph of facet-connected pivoting squares. If only pattern Γ is allowed, while patterns I and Z are forbidden, the number of connected components of \mathcal{G}_n that are singletons and the number of connected components of \mathcal{G}_n of exponential size are both exponential, regardless of the set of pivoting moves used.*

Pattern I : Corridor Bottleneck

The forbidden pattern I is weaker than pattern Γ in the sense that it suffices to make a configuration rigid for the sets of moves 1 and 2 (restrictive and leapfrog) but, if the entire set 3 of moves is allowed, pattern I alone cannot make a configuration rigid, as we will see.

► **Proposition 3.** *Let \mathcal{G}_n be the reconfiguration graph of facet-connected pivoting squares under sets 1 and 2 of pivoting moves (restrictive and leapfrog). If only pattern I is allowed, and patterns Γ and Z are forbidden, the number of connected components of \mathcal{G}_n that are singletons and the number of connected components of \mathcal{G}_n of exponential size are both exponential.*

In contrast, if the entire set of monkey-pivoting moves is allowed, then no configuration can be rigid if it only contains instances of pattern I (and no instance of patterns Γ and Z).

► **Proposition 4.** *Let \mathcal{G}_n be the reconfiguration graph of facet-connected pivoting squares under the entire set 3 of monkey-pivoting moves. If only pattern I is allowed, and patterns Γ and Z are forbidden, then \mathcal{G}_n contains no singleton components.*

Pattern Z : Wide Bottleneck

The forbidden pattern Z is weaker than the forbidden patterns Γ and I in the sense that no configuration can be rigid if it contains only instances of pattern Z.

► **Proposition 5.** *Let \mathcal{G}_n be the reconfiguration graph of facet-adjacent pivoting squares. If only pattern Z is allowed, and patterns Γ and I are forbidden, then \mathcal{G}_n contains no singleton components, regardless of the set of pivoting moves allowed.*

However, there can be locked configurations containing only instances of pattern Z.

► **Proposition 6.** *Let \mathcal{G}_n be the reconfiguration graph of facet-connected pivoting squares under pivoting set of moves 1. If only pattern Z is allowed, and patterns Γ and I are forbidden, the number of connected components of \mathcal{G}_n of exponential size is exponential.*

4 Universal Reconfiguration Algorithm with $O(1)$ Musketeers

In this section, we aim for the important practical goal of universal reconfiguration, that is, connectivity of the reconfiguration graph. We have seen that the local separation condition (while sufficient) is too strong: Robot configurations can contain many instances of the forbidden patterns and still be reconfigurable. On the other hand, we proved that as soon as the local separation condition is relaxed, the reconfiguration graph breaks into at least an exponential number of connected components of exponential size.

In what follows, we propose and analyze a new approach for reconfiguring arbitrary facet-connected configurations (which may contain an arbitrary number of instances of the forbidden patterns). Our strategy is based on the addition of $O(1)$ *musketeer modules*, i.e., modules that can freely move around the boundary of our robot configuration and will be used as helpers in certain situations. These modules are not necessarily part of the specified initial or target configuration.

4.1 Preliminaries: Outer Shell

Let C be an arbitrary facet-connected configuration of pivoting squares. We start introducing a few definitions.

Let G be the facet-adjacency graph of C , and \overline{G} the facet-adjacency graph of the lattice cells that are not occupied by a module of C . Each bounded connected component of \overline{G} is a *hole* of the robot configuration C . The only unbounded connected component of \overline{G} is the *exterior* of C . The *boundary* of C is the set of lattice cells that are empty and are facet-adjacent to (at least) one module of C . If the configuration has holes, we define its *external boundary* as the subset of the boundary contained in the unbounded connected component of \overline{G} .

► **Lemma 7.** *Let C be an arbitrary and static facet-connected configuration of pivoting squares. Let m be an active module attached to C , North of the topmost rightmost module of C . Using the monkey set of moves (set 3) m can pivot along the external boundary of C following the right-hand rule and return to its initial position. If only the leapfrog set of moves (set 2) is allowed, this is not always possible.*

The proof of this lemma is omitted due to space constraints.

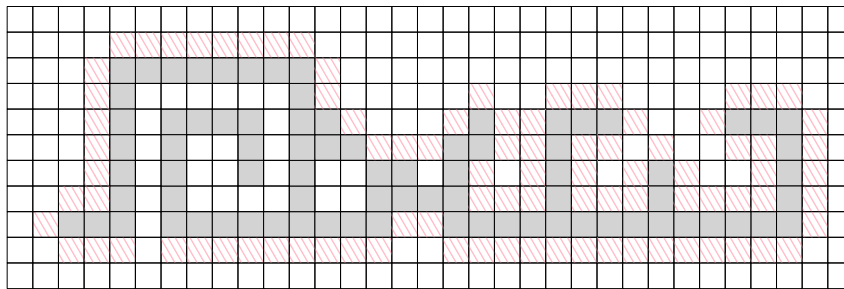
It is worth noticing that the proof of Lemma 7 does not require the use of diagonal monkey jumps, but only of straight monkey jumps. This is relevant from a practical viewpoint, since it allows our results to be applied to a larger class of modular robots. For example, the hardware systems modeled in [3, 20] can perform straight monkey jumps, but not diagonal ones.

We can now define the *outer shell* of a facet-connected configuration C of pivoting squares to be the subset of the external boundary of C formed by the lattice cells eventually occupied by any active robot module m initially positioned North of the topmost-rightmost module in C , in its right-hand rule traversal of the boundary of C , described in Lemma 7. Figure 5 illustrates this concept.

4.2 Algorithm Overview

Our reconfiguration algorithm transforms any initial facet-connected configuration C of pivoting squares into any goal configuration with the same number of modules.

In order to simplify the algorithm's description, we use an intermediate canonical configuration, say a strip, and describe the transformation from the initial shape to the strip. The reconfiguration from the strip to the final shape is obtained by reversing the steps of the



■ **Figure 5** A robot configuration (in gray) and its associated outer shell (striped in pink).

algorithm. The strip can be built from any lexicographically best positioned module of the configuration. For example, we will grow a horizontal strip to the left of the bottommost of the leftmost modules of the configuration.

The strategy behind the algorithm is simple. It consists of sequentially choosing a module from the configuration that is not a cut vertex of its facet-adjacency graph, and make it pivot, following the right-hand rule, along the outer shell, until it reaches the tip of the strip and stops. The problem of this strategy, as we saw in Section 3, is that the reconfiguration graph is not connected, even under the extended set of monkey moves. In order to overcome this problem, the algorithm uses *musketeer modules*. Any module from the canonical strip can serve as a musketeer module. We will prove that five musketeer modules are sufficient and sometimes necessary to solve any reconfiguration based on our strategy. Because the canonical strip is initially empty, it may be necessary to add musketeer modules to the strip if fewer than needed are available (this may happen at most once).

4.3 Algorithm Details

The description of the algorithm and the proof of its correctness make use of a *potential* function. If m is a module located in the lattice position with coordinates (x, y) , the potential function at m is defined as $\Phi(m) = (x + y, x)$. The potential being a two-dimensional function, we sort its values lexicographically. The maximum potential Φ_{max} (minimum potential Φ_{min}) of a configuration is the lexicographically largest (smallest) potential of all its modules. Note that, whenever we use the term *configuration*, we refer to the facet-connected component that includes all modules other than the ones in the canonical strip.

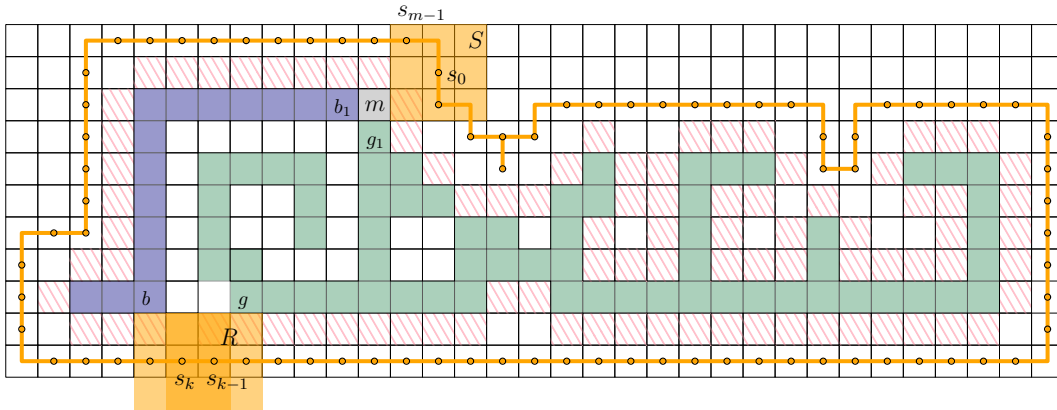
Given any configuration, we define NE and SW as being the modules with highest and lowest potential, respectively. Notice that in any configuration C , both NE and SW are facet-adjacent to the outer shell of C .

4.3.1 Musketeer Modules

► **Definition 8.** *We say that a module m is outer-free in a configuration C if it is facet-adjacent to the outer shell and it can pivot clockwise, without disconnecting the robot.*

The first step of the algorithm is to look for an outer-free module, and pivot it to the tip of the strip. This step is repeated until no further outer-free modules exist in the initial configuration. If at that point the configuration is a strip, the algorithm ends.

Otherwise, all the modules in the strip may be used as musketeer modules, one at a time, starting from the tip of the strip, pivoting them to the positions where they are needed, as described in next Section 4.3.2. Since our algorithm may require five such modules, it



■ **Figure 6** Top: 3×3 square S in its initial position s_0 . The outer thick line indicates the path traversed by the center of S . Dots correspond to the center positions where S is adjacent to a boundary edge. Bottom: the rectangular union R of S centered at s_k and at s_{k-1} .

may be necessary to add extra modules to the strip (or anywhere in the configuration where they are outer-free) in order to complete the necessary set of musketeer modules. This can be done at this stage or on the fly, as needed. This second option may be preferable in some cases, as not all configurations require as many as five musketeer modules.

4.3.2 Bridging Procedure

In this section we describe an operation necessary in some situations when there are no outer-free modules in the configuration. Let m be the NE module, i.e., the maximum potential module of a given configuration C . Trivially, there can be no modules of C located North, North-East, or East of m , i.e., in positions $(0, 1)$, $(1, 1)$, or $(1, 0)$ relative to m . Therefore, the degree of m in the facet-adjacency graph can only be 1 or 2. Since m is not outer-free, it must be a cut vertex and have degree 2. Let b_1 and g_1 respectively be its counterclockwise and clockwise facet-adjacent modules (see Figure 6). We color the two connected components of C connected by m blue and green, so that b_1 is blue and g_1 is green. One important procedure of our algorithm, which we call *bridging*, is the act of using musketeer modules to connect the green and blue components so that m becomes outer-free.

► **Observation 9.** *The outer shell has two green-blue changes of color, one happening at m .*

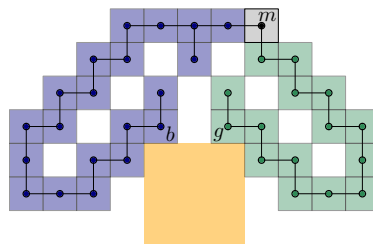
Consider a grid-aligned 3×3 square S centered at the lattice cell of coordinates $s_0 = (2, 1)$, relative to m (see top of Figure 6). Translate S orthogonally clockwise one unit at a time along the boundary of the configuration until it reaches s_0 again. Ignoring the positions where S is not adjacent to a boundary edge (i.e., the positions of S where one of its corners coincides with a convex corner of C), let s_i be the i -th position of the center of S along its boundary traversal, and let $s_m = s_0$. Refer to Figure 6. Since m is the maximum potential module of the configuration, S is empty of modules at position s_0 and all subsequent positions, and it does not share edges with the blue component when centered at s_0 , while at s_{m-2} it is facet-adjacent to the blue module b_1 . Let s_k be the first position of S along its boundary traversal where S becomes facet-adjacent to a blue module. Since S travels along the boundary of the configuration, the rectangular union R of S centered at s_k and at s_{k-1} should also be facet-adjacent to a green module (see bottom of Figure 6).

The algorithm pivots the musketeer modules clockwise, following the right-hand rule along the outer shell of the configuration, and brings them to the vicinity of s_k to connect the blue and green components, thus forming a cycle containing m .

Let g and b be the closest pair of respectively green and blue modules facet-adjacent to rectangle R , and let d be the L_1 distance between them. The bridging procedure depends on the value of d . It is easy to see that d can only be 2, 3, 4, 5, or 6. In the full version of this paper we prove the following lemma:

► **Lemma 10.** *Let m be the NE module, i.e., the maximum potential module of a given configuration C . The bridging procedure for m uses $O(n)$ pivoting operations and at most five musketeer modules, and does not change the maximum and minimum potential of the configuration. After the procedure ends, m is still the NE module of the modified configuration, but no longer a cut vertex of its facet-adjacency graph.*

Notice also that the bound of five musketeer modules for bridging is tight: Figure 7 shows an example requiring five musketeer modules for bridging.



■ **Figure 7** A rigid configuration that requires the addition of five musketeer modules for bridging.

4.3.3 Reconfiguration Step

We now need to guarantee that module m is able to move and thus it can pivot along the outer shell of C and join the canonical strip. This is clear when m is disjoint from the neighborhood of m . We also want to show that we can liberate and send to the canonical strip either the musketeers used or at least as many modules as musketeers used. In the full version of this paper we extend the analysis of the neighborhood of m , and for each of the possible cases we show that either invoking the bridging procedure or explicitly placing musketeer modules we can guarantee that.

Progress of the reconfiguration is measured in terms of the potential gap $\Delta\Phi = \Phi_{max} - \Phi_{min}$ of the configuration and the size of C (recall that C includes all modules that are not part of the canonical strip). In all the different cases we show that a reconfiguration step decreases the potential gap and/or the size of C .

4.4 Algorithm Pseudocode

Algorithm 1 solves the reconfiguration problem by combining the operations described in the previous sections:

► **Theorem 11.** *The reconfiguration algorithm (Algorithm 1) transforms a facet-connected configuration C with n modules into a canonical strip of the same size, using $O(n^2)$ monkey-move pivoting steps, which is worst-case optimal, and adding at most five extra modules.*

Proof. The input to the algorithm is a configuration C of size n and potential gap $\Delta\Phi = O(n)$. Each step of the innermost loop uses $O(n)$ pivoting operations to take an outer-free module to the end of the strip, thus decreasing the size of C by one. Each reconfiguration step uses $O(n)$ pivoting steps to decrease either the potential gap or the size of C , leaving it facet-connected

■ **Algorithm 1** Reconfiguring an arbitrary facet-connected configuration into a canonical strip.

Data: An arbitrary facet-connected configuration C with n modules
Result: A canonical strip of modules of length n

```

while there are still modules in  $C$  do
    while there exist outer-free modules do
        | pick one outer-free module and pivot it all the way to the tip of the strip;
    end
    if the strip has fewer than five modules then
        | make the strip five modules long by adding musketeer modules;
    end
    invoke the reconfiguration step;
end

```

(and never increasing the potential gap). Because the size of C never increases, the length of the canonical strip never decreases. This means that the strip can have fewer than five modules only once and the conditional does not affect the complexity of the algorithm. We conclude that the algorithm terminates after $O(n)$ iterations in total. Because each iteration takes $O(n)$ pivoting steps, the total number of pivoting steps is $O(n^2)$. Optimality comes from the $\Omega(n^2)$ pivoting steps required to reconfigure a vertical strip into a horizontal one. ◀

5 Conclusion and Open Problems

This paper addresses the problem of reconfiguring a facet-connected grid configuration of n modules into any other configuration of n modules under three increasingly more flexible sets of pivoting moves, namely restrictive, leapfrog and monkey. Previous results solve this problem under the leapfrog set of moves, as long as the initial and final configurations satisfy a strong local separating condition imposed by three forbidden patterns. We show that there exist robot configurations with many instances of the three forbidden patterns that are still reconfigurable, so the local separation condition is not necessary. On the other hand, we show that as soon as the local separation condition is relaxed, the reconfiguration graph breaks into an exponential number of connected components of exponential size. To overcome this obstacle we introduce a new pivoting move, called monkey, and a natural reconfiguration approach that does not depend on local features, but uses up to five extra modules that can freely move around the boundary of the robot configuration. These extra modules are used to unlock intermediate locked configurations so that progress can be made towards the target configuration. We show that our approach uses $O(n^2)$ monkey-pivoting moves to reconfigure any source configuration with n pivoting modules into any given target configuration.

We leave open the question of whether universal reconfiguration can be accomplished under the more restrictive set of leapfrog pivoting moves using a constant number of extra modules.

Another question is whether our approach generalizes to three or higher dimensions. For example, when the slice graphs (where the vertices are the *slices* of the configuration cut along an axis and the edges connect slices with facet-adjacent modules) of the source and target configurations are both paths, we should be able to reconfigure each to a strip of modules, one slice at a time, similar to our 2-dimensional approach does. We conjecture that a similar approach will also work for general 3-dimensional configurations, potentially after increasing the number of musketeer modules to bridge larger gaps introduced by the higher dimensionality.

References

- 1 Z. Abel and S. D. Kominers. Pushing hypercubes around. *CoRR*, abs/0802.3414, 2008. [arXiv:0802.3414](#).
- 2 B. K. An. EM-Cube: cube-shaped, self-reconfigurable robots sliding on structure surfaces. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 3149–3155, 2008.
- 3 N. Ayanian, P. J. White, Á. Hálász, M. Yim, and V. Kumar. Stochastic control for self-assembly of XBots. In *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2008.
- 4 N. M. Benbernou. *Geometric algorithms for reconfigurable structures*. PhD thesis, Massachusetts Institute of Technology, 2011.
- 5 S. Chennareddy, A. Agrawal, and A. Karupiah. Modular self-reconfigurable robotic systems: a survey on hardware architectures. *Journal of Robotics*, 2017(5013532), 2017.
- 6 G. S. Chirikjian. Kinematics of a metamorphic robotic system. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 449–455, 1994.
- 7 A. Dumitrescu and J. Pach. Pushing squares around. *Graphs and Combinatorics*, 22(1):37–50, 2006.
- 8 R. Fitch, Z. Butler, and D. Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2460–2467, 2003.
- 9 A. Hemmerling. *Labyrinth problems – labyrinth-searching abilities of automata*, volume 14 of *Teubner-Texte zur Mathematik (TTZM)*. Springer-Verlag, 1989.
- 10 H. Kurokawa, S. Murata, E. Yoshida, K. Tomita, and S. Kokaji. A 3-D self-reconfigurable structure and experiments. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 860–865, 1998.
- 11 T. Larkworthy and S. Ramamoorthy. A characterization of the reconfiguration space of self-reconfiguring robotic systems. *Robotica*, 29(1):73–85, 2011.
- 12 O. Michail, G. Skretas, and P. G. Spirakis. On the transformation capability of feasible mechanisms for programmable matter. *J. Comput. Syst. Sci.*, 102:18–39, 2019.
- 13 S. Murata, H. Kurokawa, and S. Kokaji. Self-assembling machine. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 441–448, 1994.
- 14 S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji. M-TRAN: self-reconfigurable modular robotic system. *IEEE/ASME Transactions on Mechatronics*, 7(4):431–441, 2002.
- 15 A. Nguyen, L. J. Guibas, and M. Yim. Controlled module density helps reconfiguration planning. In *Algorithmic and Computational Robotics: New Dimensions (WAFR)*, pages 23–25. A. K. Peters, 2001.
- 16 E. H. Østergaard, K. Kassow, R. Beck, and H. H. Lund. Design of the ATRON lattice-based self-reconfigurable robot. *Autonomous Robots*, 21(2):165–183, 2006.
- 17 D. Rus and M. Vona. A physical implementation of the self-reconfiguring crystalline robot. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1726–1733, 2000.
- 18 B. Salemi, M. Moll, and W.-M. Shen. SUPERBOT: a deployable, multi-functional, and modular self-reconfigurable robotic system. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3636–3641, 2006.
- 19 K. Stoy, D. Brandt, and D. J. Christensen. *Self-reconfigurable robots: an introduction*. MIT Press, 2010.
- 20 C. Sung, J. Bern, J. Romanishin, and D. Rus. Reconfiguration planning for pivoting cube modular robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1933–1940, 2015.

3:14 Universal Reconfiguration of Facet-Connected Modular Robots by Pivots

- 21 C. Unsal, H. Kiliccote, and P. Khosla. I(CES)-Cubes: a modular self-reconfigurable bipartite robotic system. In *Proc. SPIE Conference on Mobile Robots and Autonomous Systems*, volume 3839, pages 258–269. SPIE, 1999.
- 22 M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems. *IEEE Robotics & Automation Magazine*, 14(1):43–52, 2007.
- 23 V. Zykov, A. Chan, and H. Lipson. Molecubes: an open-source modular robotic kit. In *IROS-2007 Self-Reconfigurable Robotics Workshop*, 2007.

Constructing Light Spanners Deterministically in Near-Linear Time

Stephen Alstrup

University of Copenhagen, Denmark
s.alstrup@di.ku.dk

Søren Dahlgaard

SupWiz, Copenhagen, Denmark
University of Copenhagen, Denmark
soerend@di.ku.dk

Arnold Filtser

Ben Gurion University of the Negev, Israel
arnold273@gmail.com

Morten Stöckel

University of Copenhagen, Denmark
most@di.ku.dk

Christian Wulff-Nilsen

University of Copenhagen, Denmark
koolooz@di.ku.dk

Abstract

Graph spanners are well-studied and widely used both in theory and practice. In a recent breakthrough, Chechik and Wulff-Nilsen [11] improved the state-of-the-art for light spanners by constructing a $(2k - 1)(1 + \varepsilon)$ -spanner with $O(n^{1+1/k})$ edges and $O_\varepsilon(n^{1/k})$ lightness. Soon after, Filtser and Solomon [19] showed that the classic greedy spanner construction achieves the same bounds. The major drawback of the greedy spanner is its running time of $O(mn^{1+1/k})$ (which is faster than [11]). This makes the construction impractical even for graphs of moderate size. Much faster spanner constructions do exist but they only achieve lightness $\Omega_\varepsilon(kn^{1/k})$, even when randomization is used.

The contribution of this paper is deterministic spanner constructions that are fast, and achieve similar bounds as the state-of-the-art slower constructions. Our first result is an $O_\varepsilon(n^{2+1/k+\varepsilon'})$ time spanner construction which achieves the state-of-the-art bounds. Our second result is an $O_\varepsilon(m + n \log n)$ time construction of a spanner with $(2k - 1)(1 + \varepsilon)$ stretch, $O(\log k \cdot n^{1+1/k})$ edges and $O_\varepsilon(\log k \cdot n^{1/k})$ lightness. This is an exponential improvement in the dependence on k compared to the previous result with such running time. Finally, for the important special case where $k = \log n$, for every constant $\varepsilon > 0$, we provide an $O(m + n^{1+\varepsilon})$ time construction that produces an $O(\log n)$ -spanner with $O(n)$ edges and $O(1)$ lightness which is asymptotically optimal. This is the first known sub-quadratic construction of such a spanner for any $k = \omega(1)$.

To achieve our constructions, we show a novel deterministic incremental approximate distance oracle. Our new oracle is crucial in our construction, as known randomized dynamic oracles require the assumption of a non-adaptive adversary. This is a strong assumption, which has seen recent attention in prolific venues. Our new oracle allows the order of the edge insertions to not be fixed in advance, which is critical as our spanner algorithm chooses which edges to insert based on the answers to distance queries. We believe our new oracle is of independent interest.

2012 ACM Subject Classification Theory of computation \rightarrow Sparsification and spanners; Theory of computation \rightarrow Dynamic graph algorithms

Keywords and phrases Spanners, Light Spanners, Efficient Construction, Deterministic Dynamic Distance Oracle

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.4

Related Version The reader is encouraged to read the arXiv version <https://arxiv.org/abs/1709.01960>, [1].



© Stephen Alstrup, Søren Dahlgaard, Arnold Filtser, Morten Stöckel, and Christian Wulff-Nilsen; licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 4; pp. 4:1–4:15



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding *Stephen Alstrup*: Research partly supported by Innovationsfonden DK, DABAI (5153-00004A) and by VILLUM Foundation grant 16582, Basic Algorithms Research Copenhagen (BARC).

Søren Dahlgaard: Research partly supported by Advanced Grant DFF-0602-02499B from the Danish Council for Independent Research.

Arnold Filtser: Supported in part by ISF grant No. (1817/17) and by BSF grant No. 2015813.

Morten Stöckel: Research partly supported by Villum Fonden.

Christian Wulff-Nilsen: Research partly supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme.

Acknowledgements We are grateful to Michael Elkin, Ofer Neiman, and Sebastian Forster for fruitful discussions.

1 Introduction

A fundamental problem in graph data structures is compressing graphs such that certain metrics are preserved as well as possible. A popular way to achieve this is through *graph spanners*. Graph spanners are sparse subgraphs that approximately preserve pairwise shortest path distances for all vertex pairs. Formally, we say that a subgraph $H = (V, E', w)$ of an edge-weighted undirected graph $G = (V, E, w)$ is a t -*spanner* of G if for all $u, v \in V$ we have $d_H(u, v) \leq t \cdot d_G(u, v)$, where d_X is the shortest path distance function for graph X and w is the edge weight function. Under such a guarantee, we say that our graph spanner H has *stretch* t . In the following, we assume that the underlying graph G is connected; if it is not, we can consider each connected component separately when computing a spanner.

Graph spanners originate from the 80's [24, 25] and have seen applications in e.g. synchronizers [25], compact routing schemes [30, 26, 8], broadcasting [18], and distance oracles [32].

The two main measures of the sparseness of a spanner H is the size (number of edges) and the *lightness*, which is defined as the ratio $w(H)/w(MST(G))$, where $w(H)$ resp. $w(MST(G))$ is the total weight of edges in H resp. a minimum spanning tree (MST) of G . It has been established that for any positive integer k , a $(2k - 1)$ -spanner of $O(n^{1+1/k})$ edges exists for any n -vertex graph [3]. This stretch-size tradeoff is widely believed to be optimal due to a matching lower bound implied by Erdős' girth conjecture [16], and there are several papers concerned with constructing spanners efficiently that get as close as possible to this lower bound [31, 5, 28].

Obtaining spanners with small lightness (and thus total weight) is motivated by applications where edge weights denote e.g. establishing cost. The best possible total weight that can be achieved in order to ensure finite stretch is the weight of an MST, thus making the definition of lightness very natural. The size lower bound of the unweighted case provides a lower bound of $\Omega(n^{1/k})$ lightness under the girth conjecture, since H must have size and weight $\Omega(n^{1+1/k})$ while the MST has size and weight $n - 1$. Obtaining this lightness has been the subject of an active line of work [2, 7, 14, 11, 19]. Throughout this paper we say that a spanner is *optimal* when its bounds coincide asymptotically with those of the girth conjecture. Obtaining an efficient spanner construction with optimal stretch-lightness trade-off remains one of the main open questions in the field of graph spanners.

Light spanners. Historically, the main approach of obtaining a spanner of bounded lightness has been through different analyses of the classic *greedy spanner*. Given $t \geq 1$, the greedy t -spanner is constructed as follows: iterate through the edges in non-decreasing order of weight and add an edge e to the partially constructed spanner H if the shortest path distance in H between the endpoints of e is greater than t times the weight of e . The study of this

spanner algorithm dates back to the early 90's with its first analysis by Althöfer et al. [2]. They showed that this simple procedure with stretch $2k - 1$ obtains the optimal $O(n^{1+1/k})$ size, and has lightness $O(n/k)$. The algorithm was subsequently analyzed in [7, 14, 19] with stretch $(1 + \varepsilon)(2k - 1)$ for any $0 < \varepsilon < 1$. Recently, a break-through result of Chechik and Wulff-Nilsen [11] showed that a significantly more complicated spanner construction obtains nearly optimal stretch, size and lightness giving the following theorem.

► **Theorem 1** ([11]). *Let $G = (V, E, w)$ be an edge-weighted undirected n -vertex graph and let k be a positive integer. Then for any $0 < \varepsilon < 1$ there exists a $(1 + \varepsilon)(2k - 1)$ -spanner of size $O(n^{1+1/k})$ and lightness $O_\varepsilon(n^{1/k})$.¹*

Following the result of [11] it was shown by Filtser and Solomon [19] that this bound is matched by the greedy spanner. In fact, they show that the greedy spanner is *existentially optimal*, meaning that if there is a t -spanner construction achieving an upper bound $m(n, t)$ resp. $l(n, t)$ on the size resp. lightness of any n -vertex graph then this bound also holds for the greedy t -spanner. In particular, the bounds in Theorem 1 also hold for the greedy spanner.

Efficient spanners. A major drawback of the greedy spanner is its $O(m \cdot (n^{1+1/k} + n \log n))$ construction time [2]. Similarly, Chechik and Wulff-Nilsen [11] only state their construction time to be polynomial, but since they use the greedy spanner as a subroutine, it has the same drawback. Addressing this problem, Elkin and Solomon [15] considered efficient construction of light spanners. They showed how to construct a spanner with stretch $(1 + \varepsilon)(2k - 1)$, size $O_\varepsilon(kn^{1+1/k})$ and lightness $O_\varepsilon(kn^{1/k})$ in time $O(km + \min(n \log n, m\alpha(n)))$. Improving on this, a recent paper of Elkin and Neiman [13] uses similar ideas to obtain stretch $(1 + \varepsilon)(2k - 1)$, size $O(\log k \cdot n^{1+1/k})$ and lightness $O(kn^{1/k})$ in expected time $O(m + \min(n \log n, m\alpha(n)))$.

Several papers also consider efficient constructions of sparse spanners, which are not necessarily light. Baswana and Sen [5] gave a $(2k - 1)$ -spanner with $O(kn^{1+1/k})$ edges in $O(km)$ expected time. This was later derandomized by Roditty et al. [27] (while keeping the same sparsity and running time). Recently, Miller et al. [23] presented a randomized algorithm with $O(m + n \log k)$ running time and $O(\log k \cdot n^{1+1/k})$ size at the cost of a constant factor in the stretch $O(k)$.

It is worth noting that for super-constant k , none of the above spanner constructions obtain the optimal $O(n^{1+1/k})$ size or $O(n^{1/k})$ lightness even if we allow $O(k)$ stretch. If we are satisfied with nearly-quadratic running time, Elkin and Solomon [15] gave a spanner with $(1 + \varepsilon)(2k - 1)$ stretch, $O_\varepsilon(n^{1+1/k})$ size and $O_\varepsilon(kn^{1/k})$ lightness in $O(kn^{2+1/k})$ time by extending a result of Roditty and Zwick [28] who got a similar result but with unbounded lightness. However, this construction still has a factor of k too much in the lightness. Thus, the fastest known spanner construction obtaining optimal size and lightness is the classic greedy spanner – even if we allow $O(k)$ stretch or $o(kn^{1/k})$ lightness.

We would like to emphasize that the case $k = \log n$ is of special interest. This is the point on the tradeoff curve allowing spanners of linear size and constant lightness. Prior to this paper, the state of the art for efficient spanner constructions with constant lightness suffered from distortion at least $O(\log^2 n)$. See the discussion after Corollary 5 for further details.

A summary of spanner algorithms can be seen in Table 1.

¹ O_ε notation hides polynomial factors in $1/\varepsilon$.

■ **Table 1** Table of related spanner constructions. In the top of the table we list non-efficient spanner constructions. In the middle we list known efficient spanner construction. In the bottom we list our contributions. Results marked * are different analyses of the greedy spanner. Results marked # are randomized. Lightness complexities marked ** appear in the full version [1] and W denotes the maximum edge weight of the input graph. The bounds hold for any constant $\varepsilon, \varepsilon' > 0$.

Stretch	Size	Lightness	Construction	Ref
$(2k - 1)$	$O(n^{1+1/k})$	$O(n/k)$	$O(mn^{1+1/k})$	[ADD+93][2]*
$(2k - 1)(1 + \varepsilon)$	$O(n^{1+1/k})$	$O(kn^{1/k})$	$O(mn^{1+1/k})$	[CDNS92][7]*
$(2k - 1)$	$O(n^{1+1/k})$	$\Omega(W)$ **	$O(kn^{2+1/k})$	[RZ11][28]
$(2k - 1)$	$O(kn^{1+1/k})$	$\Omega(n^{1+1/k})$ **	$O(kmn^{1/k})$	[TZ05][31]#
$(2k - 1)(1 + \varepsilon)$	$O(n^{1+1/k})$	$O(kn^{1/k})$	$O(kn^{2+1/k})$	[ES16][15]
$(2k - 1)(1 + \varepsilon)$	$O(n^{1+1/k})$	$O(n^{1/k} \cdot k / \log k)$	$O(mn^{1+1/k})$	[ENS15][14]*
$(2k - 1)(1 + \varepsilon)$	$O(n^{1+1/k})$	$O(n^{1/k})$	$n^{\Theta(1)}$	[CW18][11]
$(2k - 1)(1 + \varepsilon)$	$O(n^{1+1/k})$	$O(n^{1/k})$	$O(mn^{1+1/k})$	[FS16][19]*
$(2k - 1)$	$O(kn^{1+1/k})$	$\Omega(W)$ **	$O(km)$	[BS07,RTZ05][5, 27]
$(2k - 1)(1 + \varepsilon)$	$O(kn^{1+1/k})$	$O(kn^{1/k})$	$O(km + n \log n)$	[ES16][15]
$O(k)$	$O(\log k \cdot n^{1+1/k})$	$\Omega(W)$	$O(m + n \cdot \log k)$	[MPVX15][23]#
$(2k - 1)(1 + \varepsilon)$	$O(\log k \cdot n^{1+1/k})$	$O(k \cdot n^{1/k})$	$O(m + n \cdot \log n)$	[EN17][13]#
$(2k - 1)(1 + \varepsilon)$	$O(\log k \cdot n^{1+1/k})$	$\Omega(W)$	$O(m)$	Theorem 7
$(2k - 1)(1 + \varepsilon)$	$O(\log k \cdot n^{1+1/k})$	$O(\log k \cdot n^{1/k})$	$O(m + n \cdot \log n)$	Theorem 3
$(2k - 1)(1 + \varepsilon)$	$O(n^{1+1/k})$	$O(n^{1/k})$	$O(n^{2+1/k+\varepsilon'})$	Theorem 2
$O(k)$	$O(n^{1+1/k})$	$O(n^{1/k})$	$O(m + n^{1+\varepsilon'+1/k})$	Theorem 4
$O(\log n)/\delta$	$O(n)$	$1 + \delta$	$O(m + n^{1+\varepsilon'})$	Corollary 5

1.1 Our results

We present the first spanner obtaining the same near-optimal guarantees as the greedy spanner in significantly faster time by obtaining a $(1 + \varepsilon)(2k - 1)$ spanner with optimal size and lightness in $O_\varepsilon(n^{2+1/k+\varepsilon'})$ time. We also present a variant of this spanner, improving the running time to $O(m + n \log n)$ by paying a $\log k$ factor in the size and lightness. Finally, we present an optimal $O_\varepsilon(\log n)$ -spanner which can be constructed in $O(m + n^{1+\varepsilon})$ time. This special case is of particular interest in the literature (see e.g. [4, 22]). Furthermore, all of our constructions are deterministic, giving the first subquadratic deterministic construction without the additional dependence on k in the size of the spanner. As an important tool, we introduce a new deterministic approximate incremental distance oracle which works in near-linear time for maintaining small distances approximately. We believe this result is of independent interest.

More precisely, we show the following theorems.

► **Theorem 2.** *Given a weighted undirected graph $G = (V, E, w)$ with m edges and n vertices, any positive integer k , and $\varepsilon, \varepsilon' > 0$ where ε arbitrarily close to 0 and ε' is a constant, one can deterministically construct an $(1 + \varepsilon)(2k - 1)$ -spanner of G with $O_\varepsilon(n^{1+1/k})$ edges and lightness $O_\varepsilon(n^{1/k})$ in $O(n^{2+1/k+\varepsilon'})$ time.*

► **Theorem 3.** *Given a weighted undirected graph $G = (V, E, w)$ with m edges and n vertices, a positive integer $k \geq 640$, and $\varepsilon > 0$, one can deterministically construct a $(2k - 1)(1 + \varepsilon)$ -spanner of G with $O_\varepsilon(\log k \cdot n^{1+1/k})$ edges and lightness $O_\varepsilon(\log k \cdot n^{1/k})$ in time $O(m + n \log n)$.*

Note that in Theorem 3 we require k to be larger than 640. This is not a significant limitation, as for $k = O(1)$ [15] is already optimal.

Our $O(\log n)$ -spanner is obtained as a corollary of the following more general result.

► **Theorem 4.** *Given a weighted undirected graph $G = (V, E, w)$ with m edges and n vertices, any positive integer k and constant $\varepsilon' > 0$, one can deterministically construct an $O(k)$ -spanner of G with $O(n^{1+1/k})$ edges and lightness $O(n^{1/k})$ in $O(m + n^{1+\varepsilon'+1/k})$ time.*

We note that the stretch $O(k)$ of Theorem 4 (and Corollary 5 below) hides an exponential factor in $1/\varepsilon'$, thus we only note the result for constant ε' . Bartal et. al. [4] showed that given a spanner construction that for every n -vertex weighted graph produces a $t(n)$ -stretch spanner with $m(n, t)$ edge and $l(n, t)$ lightness in $T(n, m)$ time, then for every parameter $0 < \delta < 1$ and every graph G , one can construct a t/δ -spanner with $m(n, t)$ edges and $1 + \delta \cdot l(n, t)$ lightness in $T(n, m) + (m)$ time. Plugging $k = \log n$ and using this reduction we get

► **Corollary 5.** *Let $G = (V, E, w)$ be a weighted undirected n -vertex graph, let $\varepsilon' > 0$ be a constant and $\delta > 0$ be a parameter arbitrarily close to 0. Then one can construct a spanner of G with:*

1. $O(\log n)/\delta$ stretch, $O(n)$ edges and $1 + \delta$ lightness in time $O(m + n^{1+\varepsilon'})$.
2. $O(\log n \log \log n)/\delta$ stretch, $O(n \log \log n)$ edges and $1 + \delta$ lightness in time $O(m + n \log n)$.

Corollary 5 above should be compared to previous attempts to efficiently construct a spanner with constant lightness. Although not stated explicitly, the state of the art algorithms of [15, 13], combined with the lemma from [4], provide an efficient spanner construction with $1 + \delta$ lightness, $O(n \log \log n)$ edges and only $O(\log^2 n/\delta)$ stretch.

We emphasize, that Corollary 5 is the first sub-quadratic construction of spanner with optimal size and lightness for any non-constant k .

In order to obtain Theorem 4 we construct the following deterministic incremental approximate distance oracle with near-linear total update time for maintaining small distances. We believe this result is of independent interest, and discuss it in more detail in the related work section below and in Section 3.

► **Theorem 6.** *Let G be a graph that undergoes a sequence of m edge insertions. For any constant $\varepsilon' > 0$ and parameter $d \geq 1$ there exists a data structure which processes the m insertions in total time $O(m^{1+\varepsilon'} \cdot d)$ and can answer queries at any point in the sequence of the following form. Given a pair of nodes u, v , the oracle gives, in $O(1)$ time, an estimate $\hat{d}(u, v)$ such that $\hat{d}(u, v) \geq d(u, v)$ and if $d(u, v) \leq d$ then $\hat{d}(u, v) = O(1) \cdot d(u, v)$.*

Theorem 6 assumes that ε' is constant; the O -notation hides a factor exponential in $1/\varepsilon'$ for both total update time and stretch whereas the query time bound only hides a factor of $1/\varepsilon'$.

We also obtain the following sparse, but not necessarily light, spanner in linear time as a subroutine in proving Theorem 3.

► **Theorem 7.** *Given a weighted undirected graph $G = (V, E, w)$ with m edges and n vertices, a positive integer k , and $\varepsilon > 0$, one can deterministically construct a $(2k - 1)(1 + \varepsilon)$ -spanner of G with $O_\varepsilon(n^{1+1/k} \cdot \log k)$ edges in time $O(m)$.*

1.2 Related work

Closely related to graph spanners are *approximate distance oracles (ADOs)*. An ADO is a data structure which, after preprocessing a graph G , is able to answer distance queries approximately. Distance oracles are studied extensively in the literature (see e.g. [31, 33, 9, 10]) and often use spanners as a building block. The state of the art static distance oracle

is due to Chechik [10], where a construction of space $O(n^{1+1/k})$, stretch $2k - 1$, and query time $O(1)$ is given. Our distance oracle of Theorem 6 should be compared to the result of Henzinger, et al. [21], who gave a deterministic construction for incremental (or decremental) graphs with a total update time of $O_\varepsilon(mn \log n)$, a query time of $O(\log \log n)$ and stretch $1 + \varepsilon$. For our particular application, we require near-linear total update time and only good stretch for short distances, which are commonly the most troublesome when constructing spanners. It should be added that Henzinger et al. give a general deterministic data structure for choosing centers, i.e., vertices which are roots of shortest path trees maintained by the data structure. While this data structure may be fast when the total number of centers is small, we need roughly n centers and it is not clear how this number can be reduced. Having this many centers requires at least order mn time with their data structure.

To achieve our fast update time bound, we are interested in trading worse stretch for distances above parameter d for construction time. Roditty and Zwick [29] gave a randomized distance oracle for this case, however their construction does not work against an adaptive adversary as is required for our application, where the edges to be inserted are determined by the output to the queries of the oracle (see Section 3 for more discussion on this). Removing the assumption of a non-adaptive adversary in dynamic graph algorithms has seen recent attention at prestigious venues, e.g. [34, 6]. Our new incremental approximate distance oracle for short distances given in Theorem 6 is deterministic and thus is robust against such an adversary, and we believe it may be of independent interest as a building block in deterministic dynamic graph algorithms.

For unweighted graphs, there is a folklore spanner construction by Halperin and Zwick [20] which is optimal on all parameters. The construction time is $O(m)$, it has $O(n^{1+1/k})$ edges and $2k - 1$ stretch. In Section 6 we will use this spanner as a building block in proving Theorem 3.

2 Preliminaries

Consider a weighted graph $G = (V, E, w)$, we will abuse notation and refer to as E both a set of edges and the graph itself. d_G will denote the shortest path metric (that is $d_G(v, u)$ is the weight of the lightest path between v, u in G). Given a subset V' of V , $G[V']$ is the induced graph by V' . That is it has V' as its vertices, $E \cap \binom{V'}{2}$ as its edges and w as weight function. The *diameter* of a vertex set V' in a graph G' $\text{diam}_{G'}(V') = \max_{u, v \in V'} d_{G'}(u, v)$ is the maximal distance between two vertices in V' under the shortest path metric induced by G' . For a set of edges A with weight function w , the *aspect ratio* of A is $\max_{e \in A} w(e) / \min_{e \in A} w(e)$. The *sparsity* of A is simply $|A|$ its size.

We will assume that $k = O(\log n)$ as the guarantee for lightness and sparsity will not be improved by picking larger k . Instead of proving $(1 + \varepsilon)(2k - 1)$ bound on stretch, we will prove only $(1 + O(\varepsilon))(2k - 1)$ bound. This is good enough, as Post factum we can scale ε accordingly. By O_ε we denote asymptotic notation which hides polynomial factors of $1/\varepsilon$, that is $O_\varepsilon(f) = O(f) \cdot \text{poly}(\frac{1}{\varepsilon})$.

3 Paper overview

General framework. Theorems 2 to 4 are generated via a general framework. The framework is fed two algorithms for spanner constructions: A_1 , an algorithm suitable for graphs with small aspect ratio, and A_2 , an algorithm that returns a sparse spanner, but with potentially unbounded lightness. We consider a partition of the edges into groups according to their weights. For treating most of the groups we use exponentially growing clusters, partitioning

the edges according to weight. Each such group has bounded aspect ratio, and thus we can use A_1 . Due to the exponential growth rate, we show that the contribution of all the different groups is converging. Thus only the first group is significant. However, with this approach we need a special treatment for edges of small weight. This is, as using the previous approach, the number of clusters needed to treat light edges is unbounded. Nevertheless, these edges have small impact on the lightness and we may thus use algorithm A_2 , which ignores this property.

The main work in proving Theorems 2 to 4 is in designing the algorithms A_1 and A_2 described briefly below.

Approximate greedy spanner. The major time consuming ingredient of the greedy spanner algorithm is its shortest path computations. By instead considering approximate shortest path computations we significantly speed this process up. We are the first to apply this idea on general graphs, while it has previously been applied by [12, 19] on particular graph families. Specifically, we consider the following algorithm: given some parameters $t < t'$, initialize $H \leftarrow \emptyset$ and consider the edges $(u, v) \in E$ according to increasing order of weight. If $d_H(u, v) > t' \cdot w(u, v)$ the algorithm is obliged to add (u, v) to H . If $d_H(u, v) < t \cdot w(u, v)$, the algorithm is forbidden to add (u, v) to H . Otherwise, the algorithm is free to include the edge or not. As a result, we will get spanner with stretch t' , which has the same lightness and sparsity guarantees of the greedy t -spanner. Note however, that the resulting spanner is not necessarily a subgraph of any greedy spanner.

We obtain both Theorem 2 and Theorem 4 using this approach via an incremental approximate distance oracle. It is important to note that the edges inserted into H using this approach depend on the answers to the distance queries. It is therefore not possible to use approaches that do not work against an *adaptive adversary* such as the result of Roditty and Zwick [29], which is based on random sampling. Furthermore, this is the case even if we allow the spanner construction itself to be randomized. In order to obtain Theorem 2, we use our previously described framework coupled with the “approximately greedy spanner” using an incremental $(1 + \varepsilon)$ -approximate distance oracle of Henzinger et al. [21]. For Theorem 4, we present a novel incremental approximate distance oracle, which is described below. This is the main technical part of the paper and we believe that it may be of independent interest.

Deterministic distance oracle. The main technical contribution of the paper and key ingredient in proving Theorem 4 is our new deterministic incremental approximate distance oracle of Theorem 6. The oracle supports approximate distance queries of pairs within some distance threshold, d . In particular, we may set d to be some function of the stretch of the spanner in Theorem 4. Similar to previous work on distance oracles, we have some parameter, k , and maintain k sets of nodes $\emptyset = A_{k-1} \subseteq \dots \subseteq A_0 = V$, and for each $u \in A_i$ we maintain a ball of radius $r \leq d_i$. Here, d_i is a distance threshold depending on the parameter d and which set A_i we are considering, and r is chosen such that the total degree of nodes in the ball of radius r from u is relatively small. The implementation of each ball can be thought of as an incremental Even-Shiloach tree. The set A_{i+1} is then chosen as a maximal set of nodes with disjoint balls. Here we use the fact that the vertices in A_{i+1} are centers of disjoint balls in A_i to argue that A_{i+1} is much smaller than A_i . The decrease in size of A_{i+1} pays for an increase in the maximum ball radius d_i at each level. The ball of a node u may grow in size during edge insertions. In this case, we freeze the ball associated with u , shrink the radius r associated with u , and create a new ball with the new radius. Thus, for each A_i we end up with $O(\log d)$ different radii for which we pick a maximal set of nodes with disjoint balls. For

each node $u_i \in A_i$ we may then associate a node $u_{i+1} \in A_{i+1}$ whose ball intersects with u_i 's. We use these associated nodes in the query to ensure that the path distance we find is not “too far away” from the actual shortest path distance. Consider a query pair (u, v) . Then the query algorithm iteratively finds a sequence of vertices $u = u_0 \in A_0, u_1 \in A_1, \dots, u_i \in A_i$; d_i is picked such that if v is not in the ball centered at u_i with radius d_i then the shortest path distance between u and v is at least d and the algorithm outputs ∞ . Otherwise, the algorithm uses the shortest path distances stored in the balls that it encounters to output the weight of a uv -path $(u = u_0) \rightsquigarrow u_1 \rightsquigarrow \dots \rightsquigarrow u_i \rightsquigarrow v$ as an approximation of the shortest path distance between u and v .

Almost linear spanner. Chechik and Wulff-Nilsen [11] implicitly used our general framework, but used the (time consuming) greedy spanner both as their A_2 component and as a sub-routine in A_1 . We show an efficient alternative to the algorithm of [11]. For the A_2 component we provide a novel sparse spanner construction (Theorem 7, see paragraph below). For A_1 , we perform a hierarchical clustering, while avoiding the costly exact diameter computations used in [11]. Finally, we replace the greedy spanner used as a sub-routine of [11] by an efficient spanner that exploits bounded aspect ratio (see Lemma 13). This spanner can be seen as a careful adaptation of Elkin and Solomon [15] analyzed in the case of bounded aspect ratio. The idea here is (again) a hierarchical partitioning of the vertices into clusters of exponentially increasing size. However, here the growth rate is only $(1 + \varepsilon)$. Upon each clustering we construct a super graph with clusters as vertices and graph edges from the corresponding weight scale as inter-cluster edges. To decide which edges in each scale add to our spanner, we execute the extremely efficient spanner of Halperin and Zwick [20] for unweighted graphs.

Linear time sparse spanner. As mentioned above we provide a novel sparse spanner construction as a building block in proving Theorem 3. Our construction is based on partitioning edges into $O_\varepsilon(\log k)$ “well separated” sets E_1, E_2, \dots , such that the ratio between $w(e)$ and $w(e')$ for edges $e, e' \in E_i$ is either a constant or at least k . This idea was previously employed by Elkin and Neiman [13] based on [23]. For these well-separated graphs, Elkin and Neiman used an involved clustering scheme based on growing clusters according to exponential distribution, and showed that the expected number of inter-cluster edges, in all levels combined, is small enough. We provide a linear time *deterministic* algorithm with an arguably simpler clustering scheme. Our clustering is based upon the clusters defined implicitly by the spanner for unweighted graphs of Halperin and Zwick [20]. In particular, we introduce a charging scheme, such that each edge added to our spanner is either paid for by a large cluster with many coins, or significantly contributing to reduce the number of clusters in the following level.

4 A framework for creating light spanners efficiently

In this section we describe a general framework for creating spanners, which we will use to prove our main results. The framework is inspired by a standard clustering approach (see e.g. [15] and [11]). The spanner framework takes as input two spanner algorithms for restricted graph classes, A_1 and A_2 , and produces a spanner algorithm for general graphs. The algorithm A_1 works for graphs with unit weight MST edges and small aspect ratio, and A_2 creates a small spanner with no guarantee for the lightness. The main work in showing Theorems 2, 3, and 4 is to construct the algorithms, A_1 and A_2 , that go into Lemma 8 below. The framework is described in the following lemma. The proof appears in the full version [1].

► **Lemma 8.** *Let $G = (V, E)$ be a weighted graph with n nodes and m edges and let $k > 0$ be an integer, $g > 1$ a fixed parameters and $\varepsilon > 0$. Assume that we are given two spanner construction algorithms A_1 and A_2 with the following properties:*

- A_1 computes a spanner of stretch $f_1(k)$, size $O_\varepsilon(s_1(k) \cdot n^{1+1/k})$ and lightness $O_\varepsilon(l_1(k) \cdot n^{1/k})$ in time $T_1(n, m, k)$ when given a graph with maximum weight g^k , where all MST edges have weight 1. Moreover, T_1 has the property that $\sum_{i=0}^{\infty} T_1\left(\frac{n}{g^{ik}}, m_i, k\right) = O(T_1(n, m, k))$, where $\sum_i m_i = m + O(n)$.

- A_2 computes a spanner of stretch $f_2(k)$ and size $O_\varepsilon(s_2(k) \cdot n^{1+1/k})$ in time $T_2(n, m, k)$. Then one can compute a spanner of stretch $\max((1 + \varepsilon)f_1(k), f_2(k))$, size $O_\varepsilon((s_1(k) + s_2(k))n^{1+1/k})$, and lightness $O_\varepsilon((l_1(k) + s_2(k)) \cdot n^{1/k})$ in time $O(T_1(n, m, k) + T_2(n, m, k) + m + n \log n)$.

5 Efficient approximate greedy spanner

In this section we will show how to efficiently implement algorithms A_1 and A_2 of Lemma 8 in order to obtain Theorems 2 and 4. We do this by implementing an “approximate-greedy” spanner, which uses an incremental approximate distance oracle to determine whether an edge should be added to the spanner or not.

We first prove Theorem 4 and then show in Section 5.2 how to modify the algorithm to give Theorem 2. We will use Theorem 6 as a main building block, but defer the proof of this theorem to Section 8. Our A_1 is obtained by the following lemma giving stretch $O(k)$ and optimal size $O(n^{1+1/k})$ and lightness $O(n^{1/k})$ for small weights.

► **Lemma 9.** *Let $G = (V, E, w)$ be an undirected graph with $m = |E|$ and $n = |V|$ and integer edge weights bounded from above by W . Let k be a positive integer and let $\varepsilon' > 0$ be a constant. Then one can deterministically construct an $O(k)$ -spanner of G with size $O(n^{1+1/k})$ and lightness $O(n^{1/k})$ in time $O(m + kWn^{1+1/k+\varepsilon'})$.*

We note that Lemma 9 above requires integer edge weights, but we may obtain this by simply rounding up the weight of each edge losing at most a factor of 2 in the stretch. Alternatively we can use the approach of Lemma 12 in Section 5.2 to reduce this factor of 2 to $(1 + \varepsilon)$.

Our A_2 will be obtained by the following lemma, which is essentially a modified implementation of Lemma 9.

► **Lemma 10.** *Let $G = (V, E, w)$ be an edge-weighted graph with $m = |E|$ and $n = |V|$. Let k be a positive integer and let $\varepsilon' > 0$ be a constant. Then one can deterministically construct an $O(k)$ -spanner of G with size $O(n^{1+1/k})$ in time $O(m + kn^{1+1/k+\varepsilon'})$.*

Combining Lemma 8 of Section 4 with Lemmas 9 and 10 above immediately gives us a spanner with stretch $O(k)$, size $O(n^{1+1/k})$ and lightness $O(n^{1/k})$ in time $O(m + n^{1+1/k+\varepsilon''})$ for any constant $\varepsilon'' > 0$. This is true because we may assume that $k \leq \gamma \log n$ for any constant $\gamma > 0$, and thus by picking γ and ε' accordingly we have that the running time given by Lemma 8 can be bounded by

$$O\left(m + kWn^{1+1/k+\varepsilon'} + kn^{1+1/k+\varepsilon'}\right) = O\left(m + kg^k n^{1+1/k+\varepsilon'}\right) = O\left(m + n^{1+1/k+\varepsilon''}\right).$$

5.1 Details of the almost-greedy spanner

Set $\varepsilon = 1$ ². Our algorithm for Lemma 9 is described below in Algorithm 1. It computes a spanner of stretch $c_1(1 + \varepsilon)(2k - 1)$, where $c_1 = O(1)$ is the stretch of our incremental approximate distance oracle in Theorem 6. Let $t = c_1(1 + \varepsilon)(2k - 1)$ throughout the section. The proof of Lemma 9 is postponed to the full version [1].

■ **Algorithm 1** Approximate-Greedy.

input : Graph $G = (V, E, w)$, Parameters ε, k
output : Spanner H

- 1 Create $H = (V, \emptyset)$
- 2 Initialize incremental distance oracle (Theorem 6) on H with $d = t \cdot W$
- 3 **for** $(u, v) \in E$ *in non-decreasing order* **do**
- 4 **if** $\hat{d}_H(u, v) > t \cdot w(u, v)$ **then**
- 5 Add (u, v) to H
- 6 **return** H

Next, we sketch the proof Lemma 10, by explaining how to modify the proof of Lemma 9.

Proof of Lemma 10. Recall that c_1 is defined as the constant stretch provided by Theorem 6. We use Algorithm 1 with the following modifications: (1) we pick $d = c_1(2k - 1)$, (2) when adding an edge to the distance oracle we add it as an unweighted edge, (3) we add an edge if its endpoints are not already connected by a path of at most d edges according to the approximate distance oracle.

The stretch of the spanner follows by the same stretch argument as in Lemma 9 and the fact that we consider the edges in non-decreasing order. To see that the size of the spanner is $O(n^{1+1/k})$ consider an edge (u, v) added to H by the modified algorithm. Since (u, v) was added to H we know that the distance estimate was at least $c_1(2k - 1)$. It thus follows from Theorem 6 that u and v have distance at least $2k$ in H and therefore H has girth at least $2k + 1$. It now follows that H has $O(n^{1+1/k})$ edges by a standard argument. The running time of this modified algorithm follows directly from Theorem 6. ◀

5.2 Near-quadratic time implementation

The construction of the previous section used our result from Theorem 6 to efficiently construct a spanner losing a constant factor exponential in $1/\varepsilon$ in the stretch. We may instead use the seminal result of Even and Shiloach [17] to obtain the same result with stretch $(1 + \varepsilon)(2k - 1)$ at the cost of a slower running time as detailed in Theorem 2. Below is described a version of the result of [17] which we will use.

► **Theorem 11** ([17]). *There exists a deterministic incremental APSP data structure for graphs with integer edge weights, which answers distance queries within a given threshold d in $O(1)$ time and has total update time $O(mnd)$.*

Here, the threshold means that if the distance between two nodes is at most d , the data structure outputs the exact distance and otherwise it outputs ∞ (or some other upper bound).

² In Section 5.2 we let $0 < \varepsilon < 1$ here to be arbitrary small parameter.

To obtain Theorem 2 we use the framework of Section 4. For the algorithm A_2 we may simply use the deterministic spanner construction of Roditty and Zwick [28] giving stretch $2k - 1$ and size $O(n^{1+1/k})$ in time $O(kn^{2+1/k})$. For A_1 we will show the following lemma.

► **Lemma 12.** *Let $G = (V, E, w)$ be an undirected graph with $m = |E|$ and $n = |V|$, edge weights bounded from above by W and where all MST edges have weight 1. Let k be a positive integer. Then one can deterministically construct a $(1 + \varepsilon)(2k - 1)$ -spanner of G with size $O_\varepsilon(n^{1+1/k})$ and lightness $O_\varepsilon(n^{1/k})$ in time $O_\varepsilon(m \log n + kWn^{2+1/k})$.*

Proof sketch. The final spanner will be a union of two spanners. Since Theorem 11 requires integer weights. We therefore need to treat edges with weight less than $1/\varepsilon$ separately. For these edges we use the algorithm of Roditty and Zwick [28] to produce a spanner with stretch $2k - 1$, size $O(n^{1+1/k})$ and thus total weight $O(n^{1+1/k}/\varepsilon)$.

For the remaining edges with weight at least $1/\varepsilon$ we now round up the weight to the nearest integer incurring a stretch of at most a factor of $1 + \varepsilon$. We now follow the approach of Algorithm 1 using the incremental APSP data structure of Theorem 11 and a threshold in line 4 of $(1 + \varepsilon)(2k - 1) \cdot w(u, v)$ instead. We use the distance threshold $d = (1 + \varepsilon)(2k - 1) \cdot W$.

The final spanner, H , is the union of the two spanners above. The stretch, size and lightness of the spanner follows immediately from the proof of Lemma 9. For the running time, we add in the additional time to sort the edges and query the distances to obtain a total running time of

$$O_\varepsilon(m \log n + d \cdot |E(H)| \cdot |V(H)|) = O_\varepsilon\left(m \log n + kWn^{2+1/k}\right) . \quad \blacktriangleleft$$

Now, recall that $W = g^k$, where $k \leq \log n$ and $g > 1$ is a fixed parameter of our choice. By picking g such that $g^{2k} \leq n^{\varepsilon'}$ we get a running time of $O(n^{2+1/k+\varepsilon'})$ for A_1 . Theorem 2 now follows from Lemma 8.

6 Almost Linear Spanner

Our algorithm builds on the spanner of Chechik and Wulff-Nilsen [11]. Here we first describe their algorithm and then present the modifications. Chechik and Wulff-Nilsen implicitly used our general framework, and thus provide two different algorithms A_1^{CW} and A_2^{CW} . A_2^{CW} is simply the greedy spanner algorithm.

A_1^{CW} starts by partitioning the non-MST edges into k buckets, such that the i th bucket contains all edges with weight in $[g^{i-1}, g^i)$. The algorithm is then split into k levels with the i th bucket being treated in the i th level. In the i th level, the vertices are partitioned into i -clusters, where the i -clusters refine the $(i - 1)$ -clusters. Each i -cluster has diameter $O(kg^i)$ and contains at least $\Omega(kg^i)$ vertices. This is similar to the (i, ε) -clusters in Section 4 with the modification of having two types of clusters, *heavy* and *light*. A cluster is heavy if it has many incident i -level edges and light otherwise. For a light cluster, we add all the incident i -level edges to the spanner directly. For the heavy clusters, Chechik and Wulff-Nilsen [11] create a special auxiliary cluster graph and run the greedy spanner on this to decide which edges should be added.

To bound the lightness of the constructed spanner, they show that each time a heavy cluster is constructed the number of clusters in the next level is reduced significantly. Then, using a clever potential function, they show that the contribution of all the greedy spanners is bounded. It is interesting to note, that in order to bound the weight of a single greedy spanner, they use the analysis of [14]. Implicitly, [14] showed that on graphs with $O(\text{poly}(k))$ aspect ratio, the greedy $(1 + \varepsilon)(2k - 1)$ -spanner has $O_\varepsilon(n^{1/k})$ lightness and $O(n^{1+1/k})$ edges.

4:12 Constructing Light Spanners Deterministically in Near-Linear Time

There are three time-consuming parts in [11]: 1) The clustering procedure iteratively grows the i -clusters as the union of several $(i-1)$ -clusters, but uses expensive exact diameter calculations in the original graph. 2) They employ the greedy spanner several times as a subroutine during $A_1^{C^w}$ for graphs with $O(\text{poly}(k))$ aspect ratio. 3) They use the greedy spanner as $A_2^{C^w}$.

In order to handle 1) above we will grow clusters purely based on the number of nodes in the $(i-1)$ -clusters (in similar manner to (i, ε) -clusters), thus making the clustering much more efficient without losing anything significant in the analysis. To handle 2) We will use the following lemma in place of the greedy spanner.

► **Lemma 13.** *Given a weighted undirected graph $G = (V, E, w)$ with m edges and n vertices, a positive integer k , $\varepsilon > 0$, such that all the weights are within $[a, a \cdot \Delta]$, and the MST have weight $O(na)$. One can deterministically construct a $(2k-1)(1+\varepsilon)$ -spanner of G with $O_\varepsilon(n^{1+\frac{1}{k}})$ edges and lightness $O_\varepsilon\left(n^{\frac{1}{k}} \cdot \log(\Delta)\right)$ in time $O(m + n \log n)$.*

The core of Lemma 13 already appears in [15], while here we analyze it for the special case where the aspect ratio is bounded by Δ . The main ingredient is an efficient spanner construction by Halperin and Zwick [20] for unweighted graphs. The proof of Lemma 13 is deferred to the full version [1]. Replacing the greedy spanner by Lemma 13 above is the sole reason for the additional $\log k$ factor in the lightness of Theorem 3.

Imitating the analysis of [11] with the modified ingredients, we are able to prove the following lemma, which we will use as A_1 in our framework.

► **Lemma 14.** *Given a weighted undirected graph $G = (V, E, w)$ with m edges and n vertices, a positive integer $k \geq 640$, and $\varepsilon > 0$, such that all MST edges have unit weight, and all weights bounded by g^k , one can deterministically construct a $(2k-1)(1+\varepsilon)$ -spanner of G with $O_\varepsilon(n^{1+1/k})$ edges and lightness $O_\varepsilon\left(\log k \cdot n^{\frac{1}{k}}\right)$ in time $O(m + nk)$.*

To address the third time-consuming part we instead use the algorithm of Theorem 7 as A_2 . Replacing the greedy algorithm by Theorem 7 is the sole reason for the additional $\log k$ factor in the sparsity of Theorem 3.

Combining Lemma 14, Theorem 7 and Lemma 8 we get Theorem 3. The proof of Lemma 14 deferred to the full version [1].

7 Proof of Theorem 7

The basic idea in the algorithm of Theorem 7, is to partition the edges E of G into $O_\varepsilon(\log k)$ sets E_1, E_2, \dots , such that the edges in E_i are “well separated”. That is, for every $e, e' \in E_i$, the ratio between $w(e)$ and $w(e')$ is either a constant or at least k . By hierarchical execution of a modified version of [20], with appropriate clustering, we show how to efficiently construct a spanner of size $O(n^{1+1/k})$ for each such “well separated” graph. Thus, taking the union of these spanners, Theorem 7 follows. The details appear in the full version [1].

8 Deterministic Incremental Distance Oracles for Small Distances

In this section, we present a deterministic incremental approximate distance oracle which can answer approximate distance queries between vertex pairs whose actual distance is below some threshold parameter d . This oracle will give us Theorem 6 and finish the proof of Theorem 4. In fact, we will show the following more general result. Theorem 6 follows directly by setting $k = 1/\varepsilon$ in the theorem below.

► **Theorem 15.** *Let $G = (V, E)$ be an n -vertex undirected graph that undergoes a series of edge insertions. Let G have positive integer edge weights and set $E = \emptyset$ initially. Let $\varepsilon > 0$ and positive integers k and d be given. Then a deterministic approximate distance oracle for G can be maintained under any sequence of operations consisting of edge insertions and approximate distance queries. Its total update time is $O_\varepsilon(m^{1+1/k}(3+\varepsilon)^{k-1}d(k+\log d)\log n)$ where m is the total number of edge insertions; the value of m does not need to be specified to the oracle in advance. Given a query vertex pair (u, v) , the oracle outputs in $O(k\log n)$ time an approximate distance $\tilde{d}(u, v)$ such that $\tilde{d}(u, v) \geq d(u, v)$ and such that if $d(u, v) \leq d$ then $\tilde{d}(u, v) \leq (2(3+\varepsilon)^{k-1} - 1)d(u, v)$.*

As discussed in Section 3, a main advantage of our oracle is that, unlike, e.g., the incremental oracle of Roditty and Zwick [29], it works against an adaptive adversary. Hence, the sequence of edge insertions does not need to be fixed in advance and we allow the answer to a distance query to affect the future sequence of insertions. This is crucial for our application since the sequence of edges inserted into our approximate greedy spanner depends on the answers to the distance queries.

We assume in the following that $m \geq n$; if this is not the case, we simply extend the sequence of updates with $n - m$ dummy updates. We will present an oracle satisfying Theorem 15 except that we require it to be given m in advance. An oracle without this requirement can be obtained from this as follows. Initially, an oracle is set up with $m = n$. Whenever the number of edge insertions exceeds m , m is doubled and a new oracle with this new value of m replaces the old oracle and the sequence of edge insertions for the old oracle are applied to the new oracle. By a geometric sums argument, the total update time for the final oracle dominates the time for all the previous oracles. Hence, presenting an oracle that knows m in advance suffices to show the theorem.

Before describing our oracle, we need some definitions and notation. For an edge-weighted tree T rooted at a vertex u , let $d_T(v)$ denote the distance from u to v in T , where $d_T(v) = \infty$ if $v \notin V(T)$. Let $r(T) = \max_{v \in V(T)} d_T(v)$. Given a graph H and $W \subseteq V(H)$, we let $\deg_H(W) = \sum_{v \in W} \deg_H(v)$ and given a subgraph S of H , we let $\deg_H(S) = \deg_H(V(S))$. For a vertex u in an edge-weighted graph H and a value $r \geq 0$, we let $B_H(u, r)$ denote the ball with center u and radius r in H , i.e., $B_H(u, r) = \{v \in V(H) \mid d_H(u, v) \leq r\}$. When H is clear from context, we simply write $B(u, r)$.

We use a superscript (t) to denote a dynamic object (such as a graph or edge set) or variable just after the t 'th edge insertion where $t = 0$ refers to the object prior to the first insertion and $t = m$ refers to the object after the final insertion. For instance, we refer to G just after the t 'th update as $G^{(t)}$.

In the following, let ε , k , and d be the values and let $G = (V, E)$ be the dynamic graph of Theorem 15. For each $i \in \{0, \dots, k-1\}$, define $m_i = 2m^{(i+1)/k}$ and let d_i be the smallest power of $(1+\varepsilon)$ of value at least $(3+2\varepsilon)^i d$. For each $u \in V$ and each $t \in \{0, \dots, m\}$, let $d_i^{(t)}(u)$ be the largest power of $(1+\varepsilon)$ of value at most d_i such that $\deg_{G^{(t)}}(B^{(t)}(u, d_i^{(t)}(u))) \leq m_i$. We let $B_i^{(t)}(u) = B^{(t)}(u, d_i^{(t)}(u))$ and let $T_i^{(t)}(u)$ be a shortest path tree from u in $B_i^{(t)}(u)$. Note that $T_i^{(t)}(u)$ need not be uniquely defined; in the following, when we say that a tree is equal to $T_i^{(t)}$, it means that the tree is equal to some shortest path tree from u in $B_i^{(t)}(u)$.

The data structure in the following lemma will be used as black box in our distance oracle. One of its tasks is to efficiently maintain trees $T_i^{(t)}(u)$. The proof of Lemma 16 is deferred to the full version [1].

► **Lemma 16.** *Let $U \subseteq V$ be a dynamic set with $U^{(0)} = \emptyset$ and let $i \in \{0, \dots, k-1\}$ be given. There is a deterministic dynamic data structure which supports any sequence of update operations, each of which is one of the following types:*

$\text{Insert-Edge}(u, v)$: this operation is applied whenever an edge (u, v) is inserted into E ,
 $\text{Insert-Vertex}(u)$: inserts vertex u into U .

Let t_{\max} denote the total number of operations and for each vertex u inserted into U , let t_u denote the update in which this happens. The data structure outputs in each update $t \in \{1, \dots, t_{\max}\}$ a (possibly empty) set of trees $\bar{T}_i^{(t)}(u)$ rooted at u for each $u \in U^{(t)}$ satisfying either $t > t_u$ and $d_i^{(t)}(u) < d_i^{(t-1)}(u)$ or $t = t_u$ and $d_i^{(t)}(u) < d_i$. For each such tree $\bar{T}_i^{(t)}(u)$, $r(\bar{T}_i^{(t)}(u)) \leq (1 + \varepsilon)d_i^{(t)}(u) \leq d_i$ and $\deg_{G^{(t)}}(\bar{T}_i^{(t)}(u)) > m_i$. Total update time is $O(m) + O_\varepsilon(|U^{(t_{\max})}|m_i d_i \log n)$.

At any point, the data structure supports in $O(1)$ time a query for the value $d_i^{(t)}(u)$ and in $O(\log n)$ time a query for the value $d_{T_i(u)}(v)$ and for whether $v \in V(T_i(u))$, for any query vertices $u \in U$ and $v \in V$.

The construction and analysis of the distance oracle appear in the full version [1].

References

- 1 Stephen Alstrup, Søren Dahlgaard, Arnold Filtser, Morten Stöckel, and Christian Wulff-Nilsen. Constructing Light Spanners Deterministically in Near-Linear Time. *CoRR*, abs/1709.01960, 2017. [arXiv:1709.01960](https://arxiv.org/abs/1709.01960).
- 2 Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993. doi:10.1007/BF02189308.
- 3 Baruch Awerbuch. Complexity of Network Synchronization. *J. ACM*, 32(4):804–823, October 1985. doi:10.1145/4221.4227.
- 4 Yair Bartal, Arnold Filtser, and Ofer Neiman. On notions of distortion and an almost minimum spanning tree with constant average distortion. *Journal of Computer and System Sciences*, 2019. doi:10.1016/j.jcss.2019.04.006.
- 5 Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms*, 30(4):532–563, 2007. See also ICALP’03.
- 6 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *Proc. 48th ACM Symposium on Theory of Computing (STOC)*, pages 398–411, 2016.
- 7 Barun Chandra, Gautam Das, Giri Narasimhan, and José Soares. New Sparseness Results on Graph Spanners. In *Proc. 8th ACM Symposium on Computational Geometry (SoCG)*, pages 192–201, 1992.
- 8 Shiri Chechik. Compact Routing Schemes with Improved Stretch. In *Proc. ACM Symposium on Principles of Distributed Computing (PODC)*, pages 33–41, 2013.
- 9 Shiri Chechik. Approximate Distance Oracles with Constant Query Time. In *Proc. 46th ACM Symposium on Theory of Computing (STOC)*, pages 654–663, 2014.
- 10 Shiri Chechik. Approximate Distance Oracles with Improved Bounds. In *Proc. 47th ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 2015.
- 11 Shiri Chechik and Christian Wulff-Nilsen. Near-Optimal Light Spanners. *ACM Trans. Algorithms*, 14(3):33:1–33:15, 2018. doi:10.1145/3199607.
- 12 Gautam Das and Giri Narasimhan. A Fast Algorithm for Constructing Sparse Euclidean Spanners. *Int. J. Comput. Geometry Appl.*, 7(4):297–315, 1997. doi:10.1142/S0218195997000193.
- 13 Michael Elkin and Ofer Neiman. Efficient Algorithms for Constructing Very Sparse Spanners and Emulators. *ACM Trans. Algorithms*, 15(1):4:1–4:29, 2019. doi:10.1145/3274651.
- 14 Michael Elkin, Ofer Neiman, and Shay Solomon. Light Spanners. *SIAM J. Discrete Math.*, 29(3):1312–1321, 2015. doi:10.1137/140979538.
- 15 Michael Elkin and Shay Solomon. Fast Constructions of Lightweight Spanners for General Graphs. *ACM Transactions on Algorithms*, 12(3):29:1–29:21, 2016. See also SODA’13.

- 16 Paul Erdős. Extremal problems in graph theory. In *Theory of Graphs and its Applications*, pages 29–36, 1964.
- 17 Shimon Even and Yossi Shiloach. An On-Line Edge-Deletion Problem. *J. ACM*, 28(1):1–4, 1981.
- 18 Arthur M. Farley, Andrzej Proskurowski, Daniel Zappala, and Kurt Windisch. Spanners and message distribution in networks. *Discrete Applied Mathematics*, 137(2):159–171, 2004.
- 19 Arnold Filtser and Shay Solomon. The Greedy Spanner is Existentially Optimal. In *Proc. of the 2016 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 9–17, 2016.
- 20 Shay Halperin and Uri Zwick. Linear time deterministic algorithm for computing spanners for unweighted graphs, 1996.
- 21 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Dynamic Approximate All-Pairs Shortest Paths: Breaking the $O(mn)$ Barrier and Derandomization. *SIAM Journal on Computing*, 45(3):947–1006, 2016. See also FOCS’13.
- 22 Ioannis Koutis and Shen Chen Xu. Simple Parallel and Distributed Algorithms for Spectral Graph Sparsification. *TOPC*, 3(2):14:1–14:14, 2016. doi:10.1145/2948062.
- 23 Gary L. Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. Improved Parallel Algorithms for Spanners and Hopsets. In *Proc. 27th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 192–201, 2015.
- 24 David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.
- 25 David Peleg and Jeffrey D. Ullman. An Optimal Synchronizer for the Hypercube. In *Proc. 6th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 77–85, 1987.
- 26 David Peleg and Eli Upfal. A Tradeoff Between Space and Efficiency for Routing Tables. In *Proc 20th ACM Symposium on Theory of Computing (STOC)*, pages 43–52, 1988.
- 27 Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic Constructions of Approximate Distance Oracles and Spanners. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 261–272, 2005.
- 28 Liam Roditty and Uri Zwick. On Dynamic Shortest Paths Problems. *Algorithmica*, 61(2):389–401, 2011.
- 29 Liam Roditty and Uri Zwick. Dynamic Approximate All-Pairs Shortest Paths in Undirected Graphs. *SIAM Journal on Computing*, 41(3):670–683, 2012. See also FOCS’04.
- 30 Mikkel Thorup and Uri Zwick. Compact Routing Schemes. In *Proc. 13th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 1–10, 2001.
- 31 Mikkel Thorup and Uri Zwick. Approximate Distance Oracles. *Journal of the ACM*, 52(1):1–24, January 2005. See also STOC’01. doi:10.1145/1044731.1044732.
- 32 Christian Wulff-Nilsen. Approximate Distance Oracles with Improved Preprocessing Time. In *Proc. 23rd ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 202–208, 2012.
- 33 Christian Wulff-Nilsen. Approximate Distance Oracles with Improved Query 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 539–549, 2013.
- 34 Christian Wulff-Nilsen. Fully-Dynamic Minimum Spanning Forest with Improved Worst-Case Update Time. *CoRR*, abs/1611.02864, 2016. To appear at STOC’17. arXiv:1611.02864.

Repetition Detection in a Dynamic String

Amihood Amir

Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
amir@esc.biu.ac.il

Itai Boneh

Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
barbunyaboy2@gmail.com

Panagiotis Charalampopoulos 

Department of Informatics, King's College London, London, UK
Efi Arazi School of Computer Science, The Interdisciplinary Center Herzliya, Herzliya, Israel
panagiotis.charalampopoulos@kcl.ac.uk

Eitan Kondratovsky

Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
eit414@gmail.com

Abstract

A string UU for a non-empty string U is called a square. Squares have been well-studied both from a combinatorial and an algorithmic perspective. In this paper, we are the first to consider the problem of maintaining a representation of the squares in a dynamic string S of length at most n . We present an algorithm that updates this representation in $n^{o(1)}$ time. This representation allows us to report a longest square-substring of S in $\mathcal{O}(1)$ time and all square-substrings of S in $\mathcal{O}(\text{output})$ time. We achieve this by introducing a novel tool – maintaining prefix-suffix matches of two dynamic strings.

We extend the above result to address the problem of maintaining a representation of all runs (maximal repetitions) of the string. Runs are known to capture the periodic structure of a string, and, as an application, we show that our representation of runs allows us to efficiently answer periodicity queries for substrings of a dynamic string. These queries have proven useful in static pattern matching problems and our techniques have the potential of offering solutions to these problems in a *dynamic* text setting.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases string algorithms, dynamic algorithms, squares, repetitions, runs

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.5

Funding *Amihood Amir*: Supported by Israel Science Foundation (ISF) grant 1475/18 and United States – Israel Binational Science Foundation (BSF) grant 2018141.

Panagiotis Charalampopoulos: Partially supported by Israel Science Foundation (ISF) grant 794/13.

Acknowledgements We warmly thank Tomasz Kociumaka for useful discussions.

1 Introduction

A string UU , where U is not empty, is called a *square* or a *tandem repeat*. Squares are a fundamental construct in word combinatorics, and algorithms for finding all squares have been sought as early as the 1980's [15, 10, 37]. The problem turned out to be central in computational biology causing much algorithmic work to have taken place since then [12, 27]. The approximate version is also of great interest [36, 19, 41, 40].

A run is a periodic fragment of the text that cannot be extended to either direction without increasing its period. Kolpakov and Kucherov, in their seminal paper [35], showed that there are $\mathcal{O}(n)$ runs in a text of length n , and presented an algorithm to compute them



© Amihood Amir, Itai Boneh, Panagiotis Charalampopoulos, and Eitan Kondratovsky;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 5; pp. 5:1–5:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

in $\mathcal{O}(n)$ time. After a long line of research, the breakthrough result of Bannai et al. [11] showed that a string of length n can have at most n runs. Runs have been used as an algorithmic tool, for example, for extracting the k -powers in a string (note that a square is a 2-power) and for efficient computation of the periods of substrings of a string [17, 33, 34].

Due to the importance, both theoretical and practical, of squares and runs, it is surprising that the problem of computing or maintaining them in a dynamic text has not been studied. Of course, one can re-run a square/run detection algorithm after every change in the text, but this is clearly a very inefficient way of handling the problem.

In the 1990's the active field of dynamic graph algorithms was started, with the motive of answering questions on graphs that dynamically change over time. For an overview see [18]. Recently, there has been a growing interest in dynamic pattern matching. This natural interest grew from the fact that the biggest digital library in the world - the web - is constantly changing, as well as from the fact that other big digital libraries - genomes and astrophysical data, are also subject to change through mutation and time, respectively.

Historically, there has been much interest in dynamic string matching algorithms. Amir and Farach [7] introduced dynamic dictionary matching, which was later improved by Amir et al. [8]. Idury and Scheffer [29] introduced an automaton-based dynamic dictionary algorithm. Gu et al. [26] and Sahinalp and Vishkin [39] developed a dynamic indexing algorithm, where a dynamic text is indexed. Further progress in dynamic indexing and dictionary matching was achieved by Ferragina et al. [20, 21] and Mehlhorn et al. [38]. Pattern matching algorithms where the text is dynamic and the text is static were also considered [2, 9].

In the last few years there was a resurgence of interest in dynamic string matching. In 2017 a theory began to develop with its nascent set of tools. Bille et al. [13] investigated dynamic relative compression and dynamic partial sums. Amir et al. [5] considered the longest common factor (LCF) problem in the case of one revertible edit (see also [1]). Special cases of the dynamic LCF problem were discussed by Amir and Boneh [3]. An algorithm for the fully dynamic LCF problem was presented by Amir et al. [6]. (A similar line of work has taken place for the problem of maintaining a longest palindrome in a dynamic string [23, 24, 6, 4].) Gawrychowski et al. [25] settled the complexity of maintaining a dynamic collection of strings under operations: concatenate, split, makestring, lexicographic comparison, and finding the longest common prefix of two strings.

We continue this line of work by considering squares and runs in a dynamic string. We present our algorithms for the case where the allowed update operations are substitutions. We first show our techniques in the setting of the following problem.

DYNAMIC LONGEST SQUARE

Input: A string S .

Query: For given index i (and character α), set $S[i] = \alpha$ and compute $\text{LS}(S)$.

Our contributions. We make a step forward in the exciting area of dynamic pattern matching. We give efficient dynamic solutions for a number of important problems:

1. Fully dynamic pattern matching in a text and pattern where the text length is twice the pattern length. In fact, to our knowledge, this is the first known algorithm that does not require $\Omega(\text{occ})$ time to report all pattern occurrences, i.e. it may report them in time smaller than their number, by reporting occurrences via an arithmetic progression.
2. Dynamic maintenance of the *longest* square in a text in $n^{o(1)}$ time per string update, after an $\tilde{\mathcal{O}}(n)$ -time preprocessing and using $\tilde{\mathcal{O}}(n)$ space.¹

¹ The $\tilde{\mathcal{O}}(\cdot)$ notation suppresses $\log^{O(1)} n$ factors.

3. Dynamic maintenance of all *runs* in a text within the same complexities. It is noteworthy that although a single substitution can destroy/create $\Omega(n)$ squares/runs, we can maintain a compact representation of them in subpolynomial time.
4. We conclude by showing that our representation of runs can be employed to efficiently maintain k -powers in a dynamic text and answer queries about periodicity of substrings, adapting the static solutions of Crochemore et al. [17]. Detecting internal periodicities has proven useful in several static string matching applications, thus our dynamic algorithm can potentially help the dynamic version of such problems. An overview of the literature for internal queries in static texts can be found in [32].

We introduce a *new* technique, which we expect will be a powerful tool in other dynamic string matching problems, that of dynamically maintaining *prefix-suffix matches*. This enabled us to efficiently maintain all runs in a dynamic string, which, in turn, enabled the applications presented in this paper.

2 Preliminaries

We begin with basic definitions and notation generally following [16]. Let $S = S[1]S[2] \cdots S[n]$ be a *string* of length $|S| = n$ over an integer alphabet Σ . For two positions i and j on S , we denote by $S[i..j] = S[i] \cdots S[j]$ the fragment of S that starts at position i and ends at position j (it is the empty string ε if $j < i$). A string Y , of length m with $0 < m \leq n$, is a substring of S if there exists a position i in S such that $Y = S[i..i+m-1]$. In this case we say that there exists an *occurrence* of Y in S , or, more simply, that Y *occurs in* S at (*starting*) *position* i . A substring is called *proper* if it is shorter than the whole string. A fragment $S[1..j]$, $j < n$, is called a *prefix* of S , and, analogously, a fragment $S[i..n]$, $i > 1$, is called a *suffix*. A fragment of S that is neither a prefix nor a suffix of S is called an *infix*. A string B that occurs both as a proper prefix and a proper suffix of S is called a *border* of S . A positive integer p is called a *period* of S if $S[i] = S[i+p]$ for all $i = 1, \dots, n-p$. String S has a period p if and only if it has a border of length $n-p$. We refer to the smallest period $\text{per}(S)$ of S as *the period* of the string and, analogously, to the longest border as *the border* of the string. A string S is *periodic* if $\text{per}(S) \leq |S|/2$.

By ST and S^k we denote the concatenation of strings S and T and k copies of the string S , respectively. A string of the form S^2 for some $S \in \Sigma^+$ is called a *square* and a string of the form S^k is called a *k-power*.

A *run* (also known as *maximal repetition*) is a periodic fragment $R = S[a..b]$ which cannot be extended to the left nor to the right without increasing the period $p = \text{per}(R)$, that is, $S[a-1] \neq S[a+p-1]$ and $S[b-p+1] \neq S[b+1]$. The number of runs in a string of length n is at most n [11] and all runs can be computed in $\mathcal{O}(n)$ time [35].

By $\text{lcpstring}(S, T)$ we denote the longest common prefix of S and T , by $\text{lcp}(S, T)$ we denote $|\text{lcpstring}(S, T)|$, and by $\text{lcp}(r, t)$ we denote $\text{lcp}(S[r..n], S[t..n])$. The longest common suffix lcs is defined analogously. We refer to queries returning $\text{lcp}(r, s)$ or $\text{lcs}(r, s)$ as longest common extension queries (*LCE queries*).

It is known that by maintaining Karp-Rabin fingerprints [31] for the substrings of length 2^j starting at a positions $i = 1 \pmod{2^j}$ for all $1 \leq j \leq \log n$ one can obtain the following lemma. (More involved solutions with better complexities in the \mathcal{O} -notation can be obtained by applying for instance the results of [25], cf. [6].)

► **Lemma 1.** *A dynamic string can be maintained with $\tilde{\mathcal{O}}(1)$ -time per edit operation so that LCE queries can be answered in $\tilde{\mathcal{O}}(1)$ time, using $\tilde{\mathcal{O}}(n)$ space.*

3 An $\tilde{O}(n^{2/3})$ -time algorithm

In this section we present an algorithm that reports a longest square $\text{LS}(S)$ in time $\tilde{O}(n^{2/3})$ after each substitution operation. We actually present two algorithms, one for each of the cases of $\text{LS}(S)$ being short or long. Let m , which is to be chosen later, be the distinguishing threshold between those cases. We can assume that m is a power of 2.

Main idea. In order to handle the case of $\text{LS}(S)$ being short, we split our text into $\mathcal{O}(n/m)$ overlapping fragments of length $2m$. Each substitution operation affects only two of these fragments, and we thus just recompute the longest square in them in $\mathcal{O}(m)$ time. As for the case that the $\text{LS}(S)$ UU is long, we note that the first occurrence of U must contain a fragment of length $m/4$ for some $i = 1 \pmod{m/4}$. The idea is to use such fragments as anchors and maintain all of their occurrences in the string using dynamic renaming. Then for every such fragment and every occurrence of it we would like to check whether there is any square UU that contains them “aligned” in the two occurrences of U . We show how to process fragments that have many occurrences efficiently by exploiting periodicity.

3.1 $|\text{LS}(S)| \leq m$

Preprocessing. We split the string S into overlapping fragments, each of length $2m$, starting at positions $i = 1 + j \cdot m$ for $j = 0, 1, \dots, \lceil n/m \rceil$. (Note that the last two fragments could be shorter than $2m$.) We use a linear-time algorithm ([17, 28]) to compute all squares in each of these fragments, requiring time $\mathcal{O}(m \cdot n/m) = \mathcal{O}(n)$ in total. For each fragment, we will maintain a representative longest square-substring, which is chosen arbitrarily in case of ties. We store the representatives of all fragments in a max heap, with their lengths being the keys. The max heap can be built in time $\mathcal{O}(n)$.

Query. Every position of the string is contained in at most two fragments. After each substitution operation we use a linear-time algorithm to recompute all squares in the affected fragments, requiring time $\mathcal{O}(m)$. We then update the heap in time $\mathcal{O}(\log n)$ by deleting the previous representatives (to which we have stored pointers) and inserting the new ones. We then simply retrieve the longest element in the max heap in $\mathcal{O}(1)$ time. The overall query-time complexity is $\mathcal{O}(m + \log n)$.

Correctness. The correctness of the described algorithm follows directly from the observation that each substring of S of length at most m is fully contained in at least one of the $\mathcal{O}(n/m)$ $2m$ -length fragments.

3.2 $|\text{LS}(S)| \geq m = 4k$

Let us start with an observation.

► **Observation 2.** *In a square-substring UU of S , with $|U| \geq 2k$, the first occurrence of U contains $S[i..i+k-1]$ for some $i = 1 \pmod{k}$.*

This observation guarantees that long square-substrings of S can be identified using the $\mathcal{O}(n/k)$ fragments starting at positions $i = 1 \pmod{k}$ as anchors. To this end, we maintain names for all k -length fragments of the string, such that two fragments of S have the same name if and only if they are equal. We first briefly describe the renaming technique, originating from [30], and then show how to use it in the dynamic setting.

The renaming technique. We recursively (consistently) rename pairs of letters of a string S . Let us consider the original string $S = S_0$ as the string at level 0 and the resulting string S_λ after λ iterations of renaming as the string at level λ . At level λ , the letter of S_λ at position i corresponds to $S[i..i + 2^\lambda - 1]$; in other words $S_\lambda[i] = S_\lambda[j]$ if and only if $S[i..i + 2^\lambda - 1] = S[j..j + 2^\lambda - 1]$.

Given string S_λ we rename as follows. We consider the multiset V of all pairs $p_\lambda(i) = (S_\lambda[i], S_\lambda[i + 2^\lambda - 1])$ and radix sort them in $\mathcal{O}(n)$ time. We then assign a distinct integer identifier $f(v)$ from $\{1, \dots, n\}$ to each distinct element of V . Finally, we set $S_{\lambda+1}[i] = f(p_\lambda(i))$. This process terminates after $\log n$ iterations and thus requires time $\mathcal{O}(n \log n)$ in total.

Dynamic renaming. We only maintain $\log k$ levels of renaming, i.e. strings $S_0, \dots, S_{\log k}$. We maintain the different pairs of letters at each level λ in a balanced binary search tree (bBST) B_λ ; each node of B_λ stores the name given to this pair and a counter of its occurrences in the string. We also maintain a bBST C_λ storing the letters from $\{1, \dots, n\}$ that are not currently used to rename pairs at this level. Each substitution affects at most k k -length fragments. We update their names in a bottom up manner in time $\mathcal{O}(k \log n)$ as follows. For each affected pair of letters at a level λ that changed, for example, from (a, b) to (a, c) , we search for (a, b) in B_λ and decrement its counter. In addition, if the counter reaches 0, the name given to this letter is now free and we update C_λ accordingly. We then search for (a, c) in B_λ and, (a) if we find it, we increment the counter and use the stored name, (b) else we insert (a, c) to B_λ and assign to this pair of letters the smallest element of C_λ .

In addition, for each name a of a k -length fragment (i.e. letter at level $\lambda = \log k$) we store the positions of its occurrences in S_λ in a predecessor data (bBST) structure P_a . We can perform insertions and deletions as well as perform predecessor/successor queries in P_a in $\mathcal{O}(\log n)$ time each. Below, after a brief discussion on periodicity, we will present a modification on P_a in order to compactly store the occurrences of a .

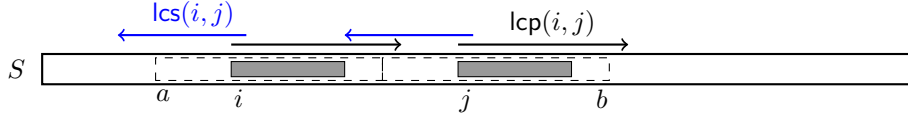
Computing squares. We would like to pair each of the k -length fragments starting at a position $i = 1 \pmod k$, with name a , with all of its other occurrences in S , which can be retrieved from the predecessor structure P_a . Let $j \neq i$ be the position of such an occurrence, and denote such a pair as (i, j) . We can assume without loss of generality that $i < j$; the other case is symmetric. For each pair, we want to check whether a square $S[a..b] = UU$, such that $a \leq i < j \leq b$ and $j - i = |U|$, exists. We call each such square an (i, j) -square. Observation 2 guarantees that every square of length at least $m = 4k$ will be identified in this manner. The following lemma shows how to perform the described check efficiently.

► **Lemma 3.** *Given two positions $i < j$, we can check whether an (i, j) -square exists and report all (i, j) -squares compactly in time $\tilde{\mathcal{O}}(1)$.*

Proof. The following observation essentially reduces computing all (i, j) -squares to answering two LCE queries. Inspect Figure 1 for an illustration.

► **Observation 4.** *An (i, j) -square UU , where i is the t -th letter of the first occurrence of U exists if and only if $\text{lcs}(i, j) \geq t$ and $\text{lcp}(i, j) \geq |U| - t + 1$.*

Now $1 \leq t \leq |U|$ and $t = i - a + 1$, where a is the starting position of such a square. Hence $a = i + 1 - t$ for $1 \leq t \leq |U|$ such that $\text{lcs}(i, j) \geq t$ and $\text{lcp}(i, j) \geq |U| - t + 1$ are the starting positions of all (i, j) -squares. Equivalently, the (i, j) -squares are the fragments $S[a..a + 2|U| - 1]$, for $i + 1 - \min\{\text{lcs}(i, j), |U|\} \leq a \leq \min\{i + \text{lcp}(i, j) - |U|, i\}$. We employ Lemma 1 to efficiently answer LCE queries. ◀



■ **Figure 1** The setting in the proof of Lemma 3. The two occurrences of U in an (i, j) -square UU are denoted by dashed rectangles. The two equal k -length fragments starting at positions i and j are denoted by gray rectangles.

Aperiodic k -length substrings. If the k -length fragment $S[i..i+k-1]$ to be processed is aperiodic, it occurs $\mathcal{O}(n/k)$ times in S . We can thus afford to employ Lemma 3 for each pair (i, j) , where $j \neq i$ is a position where $S[i..i+k-1]$ occurs. The time required to process $S[i..i+k-1]$ is thus $\tilde{\mathcal{O}}(n/k)$.

Periodic k -length substrings. If the k -length substring is periodic then we cannot process each pair individually as there could be $\Omega(n)$ of them. To overcome this, we exploit periodicity to process the pairs in batches. The lemma below follows directly from the periodicity lemma, which states that if a string has a period p and a period q , such that $p + q \leq n + \gcd(p, q)$, then $\gcd(p, q)$ is also a period of this string [22].

► **Fact 5.** *The distance between the starting positions of two consecutive occurrences of a periodic string Y with period p in a string S is either p or greater than $|Y|/2$.*

We now present an algorithm to process a k -length substring Y with period p that occurs more than $3n/k$ times in S . We can treat periodic substrings that occur fewer than $3n/k$ times with the algorithm for the aperiodic ones. Note that this is in fact necessary, as we cannot afford to compute the period of each relevant substring. Instead, we identify periodic substrings that occur frequently as follows. Remember that we have stored the positions where a k -length fragment Y with name a occurs in a predecessor data structure P_a . Then, in light of Fact 5, if Y occurs more than $3n/k$ times, two of its occurrences will have to be at distance $\text{per}(Y)$. We will identify this by checking the distance of each newly inserted element in the predecessor data structure with its predecessor and successor. If it happens to be below $|Y|/2$, we store this distance, which is $\text{per}(Y)$, as satellite information in P_a .

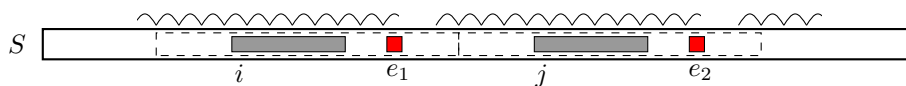
We call a set of positions $A = \{j + t \cdot p \mid t = 0, \dots, r\}$ a p -cluster of Y in S if $p = \text{per}(Y)$, $S[a..a+k-1] = Y$ for all $a \in A$ and $S[j-p..j-p+k-1] \neq Y \neq S[j+(t+1)p..j+(t+1)p+k-1]$. It follows directly from Fact 5 that there are $\mathcal{O}(n/k)$ p -clusters of Y in S . We maintain these p -clusters by storing p -cluster A as an arithmetic progression $(\min A, p, |A|)$ with key $\min A$ in P_a . We merge p -clusters if needed by using predecessor/successor queries in P_a upon insertions, and similarly split p -clusters if needed upon deletions.

► **Observation 6.** *Let Y be a periodic string. An occurrence of Y in S is a fragment of exactly one run R with $\text{per}(R) = \text{per}(Y)$. We say that R extends Y . The p -cluster containing this occurrence of Y corresponds to the occurrences of Y in R .*

► **Lemma 7.** *Given a periodic fragment Y and $p = \text{per}(Y)$, the run R that extends Y can be computed using a constant number of LCE queries. $R = S[i-a+1..i+p+b-1]$, where $a = \text{lcs}(i, i+p)$ and $b = \text{lcp}(i, i+p)$.*

We next show how to process the pairs yielded by each of the p -clusters in $\tilde{\mathcal{O}}(1)$ time.

► **Theorem 8.** *Given a position i in S , where Y occurs, and a p -cluster A of Y in S , we can compute a longest (i, j) -square over all $j \in A$ in time $\tilde{\mathcal{O}}(1)$. In particular, if $i \notin A$, we return a superset of all (i, j) -squares for $j \in A$ that are of length at least $4k$ in a compact form.*



■ **Figure 2** An illustration of the setting in Case 1 in the proof of Theorem 8. As before, the two occurrences of U in an (i, j) -square UU are denoted by dashed rectangles and the two equal k -length fragments starting at positions i and j are denoted by gray rectangles.

Proof. If it so happens that $i \in A$, then the longest (i, j) -square can be easily retrieved as it must lie entirely within the run $R = S[a..b]$ corresponding to A . Let $r = b - a \pmod{2p}$. It can be readily verified that either $S[a+r..b]$ or $S[a..b-r]$ is a longest (i, j) -square over all $j \in A$. (See also [17].)

In the other case, that is $i \notin A$, we first compute the unique run $R_1 = S[s_1..e_1]$ that extends the occurrence of Y at position i , and similarly the run $R_2 = S[s_2..e_2]$ corresponding to the occurrences of Y in A . This can be done in time $\tilde{O}(1)$ by performing a constant number of LCE queries, cf. Lemmas 7 and 1.

Our assumption that $i < j$ implies that $s_1 < s_2$. Let UU be an (i, j) -square with $j \in A$. We have the following cases for the occurrence of U in which $S[e_1 + 1]$ lies.

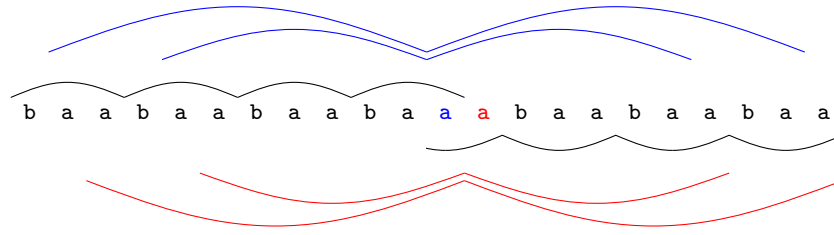
1. The first occurrence, in which case the endpoints $S[e_1]$ and $S[e_2]$ of the two runs must be aligned (i.e. be at distance $|U|$), since $\text{lcp}(i, j) > e_1 + 2 - i$. In other words, $S[e_1]$ and $S[e_2]$ must both occur as the t -th letter of an occurrence of U in the square for some t – inspect Figure 2 for an illustration. In this case we compute the longest (e_1, e_2) -square (or all (e_1, e_2) -squares) in $\tilde{O}(1)$ time using Lemma 3.
2. The second occurrence, in which case, the situation is more interesting. We have the following two subcases.
 - a. If $e_1 + 1 < s_2$, by an argument symmetric to that for the first case, the starting points $S[s_1]$ and $S[s_2]$ of the two runs must be aligned – one can think of Figure 2 reversed. As in Case 1, we can compute the longest (all) (s_1, s_2) -square(s) in $\tilde{O}(1)$ time using Lemma 3.
 - b. Else, we have that the first and second occurrences of U are fragments of runs R_1 and R_2 , respectively.

We now look into the structure yielded by the condition in Case 2b and show how to compute and represent all (possibly many) squares that satisfy it, and are essentially defined by runs R_1 and R_2 , efficiently.

► **Definition 9.** For two runs R_1 and R_2 , with period $\text{per}(R_1) = \text{per}(R_2) = p$ that overlap, we define $\text{sq}(R_1, R_2)$ to be the set of squares UU of length at least $4p$ such that the first and second occurrences of U lie entirely within R_1 and R_2 , respectively.

In what follows, we show how to compute $\text{sq}(R_1, R_2)$, which is a superset of the (i, j) -squares of length at least k for $j \in A$ since $4p \leq 4k/2 \leq 2k$. We obtain a constant number of arithmetic progressions that represent all such squares. Let us start with an example that captures the structure of $\text{sq}(R_1, R_2)$.

► **Example 10.** Consider string $(\text{baa})^4\text{a}(\text{baa})^3$. There are two runs with period $p = 3$, namely $R_1 = S[1..12]$ and $R_2 = S[12..22]$. See Figure 3 for an illustration and for the squares that satisfy the condition of Case 2b. One can see that we can get $\Omega(n)$ such squares for a string of length $\mathcal{O}(n)$, by extending this paradigm and considering string $(\text{baa})^n\text{a}(\text{baa})^n$. This example shows that a single substitution can create/destroy $\Omega(n)$ squares; think of first setting $S[n + 1] := \text{c}$ and then $S[n + 1] := \text{a}$.



■ **Figure 3** The two runs with period 3 are represented by black. The squares UU of length at least $4p$, such that the two occurrences of U are fully contained in the two runs are shown in red and blue, partitioned with respect to the first letter of the second occurrence of U .

▷ **Claim 11.** Let us suppose that we are given two runs $R_1 = S[s_1 \dots e_1]$ and $R_2 = S_2[s_2 \dots e_2]$, with $\text{per}(R_1) = \text{per}(R_2) = p$, such that $R_1[f \dots f + p - 1] = R_2[1 \dots p - 1]$ for some given $f \leq s_1 + p - 1$ and such that $s_1 \leq s_2 \leq e_1 \leq e_2$. We can compute a representation of $\text{sq}(R_1, R_2)$ in $\mathcal{O}(1)$ time.

Proof. The following fact implies that Example 10 resembles the structure of the problem.

► **Fact 12** ([32]). *Two runs with period p cannot overlap by more than $p - 1$ positions.*

Due to the condition that the first and second occurrences of U must be fragments of runs R_1 and R_2 , respectively, we have that the second occurrence of U can only start at one of the positions in $C = \{s_2, \dots, e_1 + 1\}$, where $|C| \leq p$ by Fact 12. Let us consider some $c \in C$ and characterize all squares $S[a \dots b] = UU$ with $c = a + |U|$ and $|U| \geq 2p$.

$S[c - p \dots c - 1]$ is a rotation of $S[c \dots c + p - 1]$, i.e. there exists some $\delta < p$ such that $S[c - p \dots c - 1] = S[c + \delta \dots c + p - 1]S[c \dots c + \delta - 1]$. In particular, $\delta = s_2 - f \pmod{p}$.

$|U|$ must equal $t \cdot p + \delta$ in order for the two occurrences of U to start at the same offset $\text{mod } p$ from f and s_2 ; this is necessary, since otherwise we would have two different rotations of $R_2[1 \dots p - 1]$ matching, which is impossible as it would imply that $\text{per}(R_2) < p$. In addition, all $|U|$'s of the form $t \cdot p + \delta$ for $t \geq 2$ and for which the two occurrences of U lie entirely within runs R_1 and R_2 , respectively, define valid squares. We can thus compute all these squares in $\mathcal{O}(1)$ time and represent them as an arithmetic progression with respect to $|U|$.

► **Example 13** (Continued.). For position 12 of $(baa)^4 a (baa)^3$, the blue a in Figure 3, we have $\delta = 1$ and hence the squares UU that we obtain with this as starting position of the second occurrence of U are for $|U| = 1 + 3t$, for $t = 2, 3$.

Iterating over $c \in C$ in increasing order, we only have to (a) shift all squares by 1 position each time, and (b) identify the – at most two – shifts that yield an increment/decrement in the length of the arithmetic progression due to one more/less square being allowed after the shift. We can infer the values of c for which we must increment/decrement in $\mathcal{O}(1)$ time from the endpoints of the two runs and δ . These values, p , and the arithmetic progression for $c = s_2$ are our representation of $\text{sq}(R_1, R_2)$. ◀

We can straightforwardly extract the longest (i, j) -square for $j \in A$ if it is of length at least k from this representation, and this concludes the proof of the theorem. ◀

To summarize, we spend $\tilde{\mathcal{O}}(k)$ time for the dynamic renaming and then process each of the $\mathcal{O}(n/k)$ fragments starting at positions $i = 1 \pmod{k}$ in time $\tilde{\mathcal{O}}(n/k)$, using Lemma 3 and Theorem 8. The overall time complexity of this algorithm is thus $\tilde{\mathcal{O}}(n^2/k^2 + k)$.

Wrap-up. By setting $m = 4k = n^{2/3}$ and combining the algorithms for $\text{LS}(S) \leq m$ and $\text{LS}(S) \geq m$ we obtain the following result.

► **Theorem 14.** DYNAMIC LONGEST SQUARE queries can be answered in time $\tilde{O}(n^{2/3})$, using $\tilde{O}(n)$ space, after an $\tilde{O}(n)$ -time preprocessing.

4 An $n^{o(1)}$ -time algorithm

Main Idea. If we manage to get rid of the $\mathcal{O}(m)$ time dedicated to renaming in the algorithm for computing long squares, we can then recursively obtain faster algorithms. This can be achieved by using our fastest $o(m)$ query-time dynamic algorithm for each updated $2m$ -length fragment for the case that $\text{LS}(S) \leq m$ instead of recomputing them in $\mathcal{O}(m)$ time using the static algorithm. We would then obtain a faster algorithm, and could plug this in turn for the case that $\text{LS}(S) \leq m$; and so on.

Towards the goal of getting rid of renaming, we first observe that it is wasteful to keep track of the occurrences of all k -length substrings of S . It would be sufficient to keep track of the occurrences of each k -length substring that occurs at a position $i = 1 \pmod{k}$. This could be solved by maintaining $\mathcal{O}(n/m)$ instances of dynamic pattern matching with pattern $S[i..i+k-1]$, for each $i = 1 \pmod{k}$, and text S . (Note that both the pattern and the text would have to be dynamic.) The main complication stems from the need to maintain p -clusters efficiently. To the best of our knowledge, the known pattern matching algorithms in the dynamic setting require $\Omega(\text{occ})$ time to report the occ occurrences of the pattern in the text after each update, which is unsatisfactory in our case.

2-1 Dynamic Pattern Matching. A further observation, is that we can reduce the problem to an even easier one by applying the standard trick as follows. For every substring of length k occurring at a position $i = 1 \pmod{k}$, we maintain a dynamic pattern matching instance with every substring of length $2k$ starting at a position $i = 1 \pmod{k}$. Note that every possible occurrence of the k -length fragments of interest is contained in one (and at most two) of these $2k$ -length fragments. At first glance, it may seem like this partition will be less efficient to maintain because now instead of $\mathcal{O}(n/k)$ instances of dynamic pattern matching we have $\mathcal{O}((n/k)^2)$ instances of 2-1 DYNAMIC PATTERN MATCHING – to be formally defined soon. However, this is actually lossless, since every change in S only affects $\mathcal{O}(n/k)$ such instances. Let us formally define the problem in scope.

2-1 DYNAMIC PATTERN MATCHING

Given two strings P and T with $|T| = 2|P| = 2n$, return all occurrences of P in T after each substitution operation on either of P, T .

We want to exploit the constant ratio between the lengths of the pattern and the text to obtain an efficient algorithm for 2-1 DYNAMIC PATTERN MATCHING. We further reduce this problem to another, simpler one. A partition of the text T to its n -length prefix and suffix, analogously partitions any occurrence of P at some position i . Specifically, this occurrence is partitioned to the prefix $P[1..m]$ of P , corresponding to the suffix $T[i..n]$ of $T[1..2n]$ and the suffix $P[m+1..n]$ of P , corresponding to the prefix $T[n+1..i+n-1]$ of $T[n+1..2n]$. Thus, if we know all the prefixes of P that are suffixes of $T[1..n]$, we can extend each of them in order to compute all the occurrences of P in T . (This will be a bit more involved as they will be given as arithmetic progressions, see Lemma 20.) We call a prefix of P that is a suffix of T a *prefix-suffix match of P and T* .

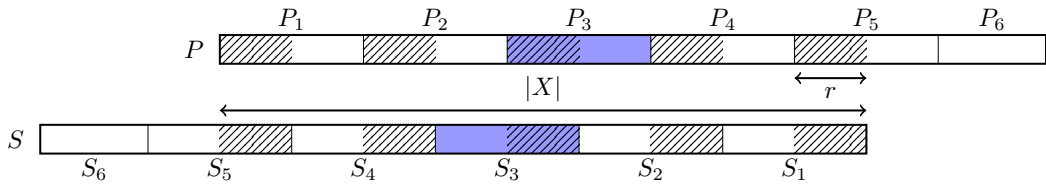
4.1 Dynamic Prefix-Suffix

We now focus on the following problem.

DYNAMIC PREFIX-SUFFIX
 Given two strings P and S of the same length n , report all the prefixes of P that are suffixes of S , after each substitution operation on either of P , S .

We partition each of P and S to $\lfloor n/m \rfloor$ m -length fragments and possibly an extra shorter fragment. Specifically, we partition P to $P_1, P_2, \dots, P_{\lfloor n/m \rfloor}$ with $P = P_1 P_2 \dots P_{\lfloor n/m \rfloor}$ and S to $S_{\lfloor n/m \rfloor}, S_{\lfloor n/m \rfloor - 1}, \dots, S_1$ with $S = S_{\lfloor n/m \rfloor} S_{\lfloor n/m \rfloor - 1} \dots S_1$. Fragments $P_{\lfloor n/m \rfloor}$ and $S_{\lfloor n/m \rfloor}$ are allowed to be of length less than m .

► **Observation 15.** *Let X be a prefix of P that is also a suffix of S . Let $x = \lceil |X|/m \rceil$ and $r = |X| \pmod m$. Every pair of fragments (P_i, S_j) that satisfies $i + j - 1 = x$, will satisfy that the prefix of length r of P_i will be equal to the suffix of the same length of S_j . (Inspect Figure 4 for an illustration.)*



■ **Figure 4** An illustration of the setting in Observation 15 with $x = 5$.

Algorithm. Relying on Observation 15, we design a recursive algorithm. For every $1 \leq x \leq \lfloor n/m \rfloor$ we will maintain an instance of **DYNAMIC PREFIX-SUFFIX** between some pair (P_i, S_j) that satisfies $i + j - 1 = x$. Namely, for a given x , we will consider the pair $(P_{\lceil y \rceil}, S_{\lfloor y \rfloor})$, where $y = (x + 1)/2$. It can be readily verified that $\lceil y \rceil + \lfloor y \rfloor - 1 = x$. Note that each P_i, S_j is in at most two of the considered pairs. Hence, each update in P or S results in no more than 2 such pairs being affected.

The prefix-suffix matches of each pair (S_i, P_j) are witnesses for possible prefix-suffix matches between P and S . All $\mathcal{O}(|S_i|) = \mathcal{O}(m)$ witnesses of a given pair can be confirmed with a logarithmic number of LCE queries, exploiting periodicity – the details are omitted due to space constraints.

We efficiently maintain the prefix-suffix matches for all relevant pairs using predecessor structures, analogously to how we maintained all starting positions of the occurrences of a substring corresponding to some name in Section 3, relying on the following lemma.

► **Lemma 16** (cf. [34, 6]). *The prefixes of a string P that are suffixes of a string S , with $|P|, |S| = \mathcal{O}(n)$, of lengths between 2^j and $2^{j+1} - 1$ form an arithmetic progression. If it has at least three elements, all these prefix-suffix matches have the same period, equal to the difference of the progression.*

Given a change, we recursively update the witnesses for the two affected pairs. At each level of the recursion, we confirm *all* witnesses. This is necessary since a witness that was not affected by the last substitution and was not an instance of a prefix-suffix match between P and S may have just become a prefix-suffix match between P and S due to the last substitution. The opposite case is possible as well.

After obtaining all the prefix-suffix matches we iterate over them to merge consistent periodic clusters as follows. For every $j \in \{1, \dots, \lceil \log n \rceil\}$, we group the prefix-suffix matches of lengths $s \in [2^{j-1}, \dots, 2^j - 1]$ and represent them as an arithmetic progression, relying on Lemma 16. The merging is necessary for the output of the algorithm to be in a compact form at every level of the recursion. It should also then be clear that the arithmetic progressions the algorithm returns are non-overlapping, as there is a unique such progression for the elements of length between 2^{j-1} and $2^j - 1$ for each j .

Complexity. The time complexity is $T(n) = 2T(m) + \tilde{O}(\lceil n/m \rceil) = 2T(\lfloor n/k \rfloor) + \tilde{O}(k)$ for $k = \lceil n/m \rceil$. $2T(m)$ for updating the two affected pairs and $\tilde{O}(n/m)$ for confirming and merging all witnesses. We omit the proof of the following fact.

► **Fact 17.** *If $T(n) = 2T(\lfloor n/k \rfloor) + c_1 k \log^{c_2} k$ for all $n \geq N_0$, where $k = \lfloor 2\sqrt{\log n} \rfloor$, c_1, c_2 are constants, and $T(C) = \mathcal{O}(1)$ for all $C = \mathcal{O}(1)$, then $T(n) = n^{o(1)}$.*

We arrive at the following theorem for DYNAMIC PREFIX-SUFFIX.

► **Theorem 18.** *A representation of all prefix-suffix matches as $\mathcal{O}(\log n)$ arithmetic progressions of their ending positions in P can be maintained with $n^{o(1)}$ time per substitution.*

By maintaining a DYNAMIC PREFIX-SUFFIX instance for $S = P$ we obtain the following corollary, as prefix-suffix matches correspond to borders of S .

► **Corollary 19.** *The period of a string $|S|$ can be maintained with $|S|^{o(1)}$ time per substitution.*

4.2 Wrap-up and complexity

The proof of the following lemma, which uses Theorem 18 as a black box, is omitted due to space constraints.

► **Lemma 20.** *2-1 DYNAMIC PATTERN MATCHING can be solved with $n^{o(1)}$ time per substitution, reporting the starting positions of all occurrences of P in T as an arithmetic progression.*

For the computation of long squares, after each substitution we proceed as follows.

1. We update each of the $\mathcal{O}(n/k)$ affected 2-1 DYNAMIC PATTERN MATCHING instances in $f(k) = k^{o(1)}$ time, employing Lemma 20.
2. We apply Lemma 3 and Theorem 8 a total of $\mathcal{O}(n^2/k^2)$ times.

The $\tilde{O}(n^2/k^2)$ term dominates the time complexity if $n/k \geq f(k)$. We note that f is an increasing function and hence it suffices to have $k \leq n/f(n)$.

Let us express the complexity of our best algorithm for DYNAMIC LONGEST SQUARE as $n^\alpha f(n) \log^\beta n$, for $\alpha < 1$ with $n^\alpha \geq (f(n))^2$ and for β being the maximum of the powers of $\log n$ hidden by the $\tilde{O}(\cdot)$ notation in Lemma 3 and Theorems 8 and 14. (Thus Theorem 14 shows an $\mathcal{O}(n^{2/3} \log^\beta n)$ -time algorithm.) Then, for $k = (n^2/f(n))^{1/(\alpha+2)}$, we have that

$$\mathcal{O}(n^2/k^2 \log^\beta n + k^\alpha f(k) \log^\beta n) = \tilde{O}(n^{2\alpha/(\alpha+2)} ((f(n))^{2/\alpha+2} + (f(n))^{\alpha+1/\alpha+2}) \log^\beta n) = \tilde{O}(n^{2\alpha/(\alpha+2)} f(n) \log^\beta n).$$

Note that this k satisfies the condition $k \leq n/f(n)$, since

$$k = (n^2/f(n))^{1/(\alpha+2)} \leq n/f(n) \iff f(n) \leq n^{\alpha/(\alpha+1)},$$

and the latter holds due to our assumptions on the value of α .

5:12 Repetition Detection in a Dynamic String

One can show by induction that $g^{(2^t)}(1) = 1/t$ for $g(x) = 2x/(2+x)$; note that $g(1) = 2/3$. We can thus construct an algorithm requiring time arbitrarily close to $\tilde{O}((f(n))^3) = n^{o(1)}$ time per update, recursively, since we can obtain an $\tilde{O}(n^{g^{t+1}(1)} f(n))$ -time algorithm by using the $\tilde{O}(n^{g^t(1)} f(n))$ -time algorithm for short squares. We thus arrive at the following result.

► **Theorem 21.** DYNAMIC LONGEST SQUARE queries can be answered in time $n^{o(1)}$, using $\tilde{O}(n)$ space, after an $\tilde{O}(n)$ -time preprocessing.

5 Maintaining all runs and applications

In this section, we first discuss how to modify the algorithm to maintain all runs instead of computing the longest square. Afterwards, by adapting the solutions of [17] for the static setting, we show several types of queries that can be answered with our representation of runs. In particular, we show how to maintain the number of all k -powers in $n^{o(1)}$ time and report the longest k -power in S for some fixed k within the same time complexity. All – possibly $\Theta(n^2)$ – k -powers can be reported in a compact way in $\tilde{O}(\text{runs})$ time, where runs denotes the number of runs in S . Finally, we show how to answer the following queries in $\tilde{O}(1)$ time: given a fragment determine if it is periodic, and, if so, compute its period.

We start by describing how to maintain all runs in the $\tilde{O}(n^{2/3})$ -time solution.

For short runs, we use the $\mathcal{O}(m)$ -time algorithm of [35]. For each $2m$ -length fragment, we only maintain runs that do not touch its endpoints, as we do not want to maintain a run that may extend to other fragments. (We only waive this restriction for runs that are suffixes/prefixes of S and are of length smaller than m .) This is sufficient as every run R such that $|R| < m$ will be fully contained in one (and at most two) of the $2m$ -length fragments. Upon a substitution we just recompute the runs for the two affected fragments.

As for runs of length at least $m = 4k$, we recompute all of them. Let us first amend Observation 2 as follows.

► **Lemma 22.** A run $R = S[a..b]$, of length at least $4k$, contains a fragment $S[i..i+k-1]$, for some $i = 1 \pmod{k}$, that also occurs at position $i + \text{per}(R)$.

Proof. The first $2k$ -length fragment of the run must appear again somewhere in the run (otherwise it is not even a square). This fragment, being of length $2k$, must contain a fragment $S[i..i+k-1]$ with $i = 0 \pmod{k}$ and $i \leq a+k-1$. $S[i..i+k-1]$ will certainly occur at position $i + \text{per}(R)$, since $i + \text{per}(R) + k - 1 \leq a+k-1 + |R|/2 + k - 1 \leq a + |R|/4 - 1 + |R|/2 + |R|/4 - 1 \leq b$. ◀

We then proceed as in Section 3. We first define the (i,j) -run to be the unique run R containing $S[i..j]$, in which the difference between i and j is consistent with the period of the run; formally, $j = i \pmod{p}$, where $p = \text{per}(R)$. Now observe that every run R that is longer than $4k$ is an (i,j) -run for some $i = 0 \pmod{k}$ and some j for which $S[i..i+k] = S[j..j+k]$ due to Lemma 22; in particular, the smallest such j is $j = i + \text{per}(R)$.

Observation 4 can be modified analogously as follows.

► **Observation 23.** An (i,j) -run R , exists if and only if $\text{lcs}(i,j) + \text{lcp}(i,j) \geq |j-i| + 1$. If R exists it is $S[i - \text{lcs}(i,j) + 1..j + \text{lcp}(i,j) - 1]$.

The above observation allows us to efficiently process names with less than $3n/k$ occurrences in S . As for names corresponding to k -length substrings with more occurrences, the proof of Theorem 8 shows that we can process the k -length substring $S[i..i+k-1]$ with a

p -cluster A of its occurrences as follows. If $i \in A$ we are done as we simply report the run with period p corresponding to A . Otherwise, using the notation of the proof of Theorem 8, we compute the (e_1, e_2) -run and the (s_1, s_2) -run if we are in Case 1 or 2a.

Note that a run $S[a..b]$ may be identified multiple times. We remove duplicates and store $S[a..b]$ as (a, b, p) , where p is the minimum $j - i$ for which this run was obtained as the (i, j) -run; by Lemma 22 we have that $p = \text{per}(S[a..b])$.

Case 2b is again more tricky. We adapt our solution for squares (see Theorem 8) in order to compute and maintain such runs compactly using arithmetic progressions.

► **Observation 24.** *All elements of $\text{sq}(R_1, R_2)$ of equal length are extended by the same run.*

We denote the elements of $\text{sq}(R_1, R_2)$ of length $2|U|$ by $G_{|U|}$.

► **Lemma 25.** *If the minimum starting position among the elements of $G_{|U|}$ is $u > s_1$ and the maximum ending position is $v < e_2$, then the run extending the elements of $G_{|U|}$ is $R = S[u..v]$ and $\text{per}(R) = |U|$.*

Proof. We have $u + |U| = s_2$ since $u > s_1$. If the run extending the squares of $G_{|U|}$ started at some position smaller than u , this would imply $S[u - 1..s_2 - 2] = S[s_2 - 1..s_2 + |U| - 2]$, which in turn would imply that the right hand side of the equation is a string with period p . This would contradict R_2 being a run. The argument for the other side is symmetric.

As for arguing that $\text{per}(R) = |U|$, let us assume for the sake of contradiction that it has a period $q < |U|$. Then, as $|U|$ is also a period of R , the periodicity lemma implies that $q' = \text{gcd}(|U|, q) \leq |U|/2$ is also a period of R .

We can apply the periodicity lemma again, since p is also a period of U and $p + q' \leq |U|$. We then have that $p' = \text{gcd}(p, q') < |p|/2$ is a period of $S[s_2..s_2 + |U| - 1]$ and a divisor of p . This is a contradiction as $S[s_2..s_2 + p - 1]$ is a primitive string, i.e. is not of the form T^k for a string T and $k > 1$, since otherwise $p' < p$ would also be a period of R_2 . ◀

We maintain all such runs $\text{runs}(R_1, R_2)$ compactly as a constant number of arithmetic progressions with respect to $|U|$; one for each of the at most three distinct group sizes.

Only two groups of squares may contain a square that starts in the first position of R_1 or ends in the last position of R_2 . These groups of squares are the only ones such that the run extending them may not be fully contained in $S[s_1..e_2]$. This could be the case for example if we had a run R_3 with the same period and appropriate overlap with R_2 . We compute the run extending a square of each of the at most two relevant groups using LCE queries and maintain these runs explicitly.

All (explicitly or compactly represented) runs are stored in a way that allows for efficient deletion, using a key with respect to their origin, i.e. the substring at some level of the recursion for which they were computed. After each substitution, the algorithm computes all the relevant runs for the substring that contains the updated position at each level of the recursion from scratch. Thus, for every substring for which we recompute runs, we first delete all runs that have this substring as key.

► **Theorem 26.** *We can maintain all runs $R = S[a..b]$ of a string of length n , as triplets $(a, b, \text{per}(R))$ and arithmetic progressions with $n^{o(1)}$ time per operation, using $\tilde{O}(n)$ space, after $\tilde{O}(n)$ -time preprocessing.*

5.1 Application I: k -powers

Let us recall that a k -power is a string of the form U^k for some non-empty string U . The authors of [17] show that given a run R as (a, b, p) , one can compute in $\mathcal{O}(1)$ time:

1. a longest substring U^k of R with $\text{per}(R) \mid |U|$;
2. the number of all fragments $S[a..b] = V^k$, with $\text{per}(R) \mid |V|$, that lie entirely within R .

For any fixed $k \geq 3$, we can maintain the longest k -power by storing a heap keeping the longest that each explicitly stored run contributes and maintain the count on the number of k -powers (not distinct) in S within the time complexities of Theorem 26. Note that by Lemma 25 and the fact that $v - u \leq 2|U| + p$ and $|U| \geq 2p$, where u, v, p and $|U|$ are as in the statement of that lemma, we have that the runs stored as arithmetic progressions do not contribute any k -powers for $k \geq 3$. Finally, we can extract all – possibly $\Theta(n^2)$ – non-distinct k -powers in $\tilde{\mathcal{O}}(\text{runs})$ time in a compact form from the runs [17].

► **Remark 27.** As for maintaining the $\mathcal{O}(n)$ distinct k -powers efficiently, we should first be able to group runs by their Lyndon roots (the Lyndon root of a run R is the lexicographically smallest rotation of a $\text{per}(R)$ -length substring of R). It is not clear how to amend our solution to maintain the runs in this way.

5.2 Application II: 2-Period Queries

2-PERIOD QUERIES

Given a fragment $S[i..j]$ of S , decide whether $S[i..j]$ is periodic and, if so, compute its period.

2-Period Queries in a static string. 2-PERIOD QUERIES were introduced in [17], while general internal period queries were introduced in [33]. The authors of [34] showed how to optimally answer 2-PERIOD QUERIES in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$ -time preprocessing. In these works, it is shown, that in order to answer the query for $S[i..j]$ it suffices to find the run R that extends $S[i..j]$, or conclude that there is no such run. In other words, it suffices to find the run R with the smallest period among the runs fully containing $S[i..j]$. Then, if $\text{per}(R) < (j - i)/2$, the fragment is periodic with period $\text{per}(R)$ and otherwise it is not.

To the best of our knowledge there is no prior work on answering internal period queries in a dynamic string. In what follows we sketch the proof of the following result – the details are omitted due to space constraints.

► **Theorem 28.** 2-PERIOD QUERIES can be answered in $\tilde{\mathcal{O}}(1)$ time in a string S of length n , with each substitution operation processed in time $n^{o(1)}$, after an $\tilde{\mathcal{O}}(n)$ -time preprocessing. The required space is $\tilde{\mathcal{O}}(n)$.

In order to compute the run with the smallest period that contains $S[i..j]$, the authors of [17] show that it is enough to be able to answer orthogonal range minimum queries in 2-d, over the following collection of points: for each run (a, b, p) we have point (a, b) with weight p . The desired run then corresponds to the point with minimum weight in the rectangle $[1, i] \times [j, n]$. A restricted version of the main result of [14], is that one can maintain a collection of $\mathcal{O}(n)$ points in $[n]^d$, for any constant d , with $\tilde{\mathcal{O}}(1)$ time per update, such that orthogonal range emptiness queries can be answered in $\tilde{\mathcal{O}}(1)$ time. We note that 2-d orthogonal range minimum queries reduce to 3-d orthogonal range emptiness queries via binary search. We maintain this data structure over the runs that are maintained explicitly, see Theorem 26. The above discussion covers the case that the run extending $S[i..j]$ has

been stored explicitly. In particular, following our discussion in Section 5.1, we are already able to answer 3-PERIOD QUERIES, i.e. whether a substring $S[i..j]$ has period at most $\lfloor j-i \rfloor/3$, and if so, return this period.

As for 2-PERIOD QUERIES, we now provide the intuition for handling the case that the run of minimum period that contains $S[i..j]$ is stored implicitly. We want to check all runs extending some square $UU \in \text{sq}(R_1, R_2)$ that is a prefix of $S[i..j]$, for some runs R_1, R_2 . Note that our definition of $\text{sq}(R_1, R_2)$ implies that $\text{per}(U) = \text{per}(R_1) \leq |U|/2$; i.e. R_1 extends U . The following lemma implies that we can only have a logarithmic number of such squares.

► **Lemma 29** ([32, Corollary 5.1.3]). *Let U_1, U_2, U_3 be periodic fragments of a text T , all starting at the same position, and being extended by runs R_1, R_2 and R_3 , respectively. If $\lfloor \log |U_1| \rfloor = \lfloor \log |U_2| \rfloor = \lfloor \log |U_3| \rfloor$, then the three runs R_1, R_2 and R_3 cannot be all distinct.*

For every set $\text{runs}(R_1, R_2)$, we add the point (s_1, e_2, p) in an initially empty 3-d grid – we use the same notation as above. We report all relevant points using 3-d dynamic orthogonal range reporting queries, again employing [14]. In particular, we first retrieve the points in the range $[1, i] \times [j, n] \times [1, \lfloor j-i \rfloor/4]$. There are $\mathcal{O}(\log n)$ of them due to the above lemma. Then, for each point, corresponding say to $\text{runs}(R_1, R_2)$, we compute in $\tilde{\mathcal{O}}(1)$ time the run of smallest period in $\text{runs}(R_1, R_2)$ containing $S[i..j]$. In particular it is the run of minimum length in $\text{runs}(R_1, R_2)$ containing $S[i..j]$ by Lemma 25.

6 Concluding remarks

We believe that, with due care, our algorithm can be adapted to handle insertions and deletions – the details are omitted due to space constraints. We leave open the questions of whether the runs of a string (or other information sufficient for answering 2-PERIOD QUERIES in $\tilde{\mathcal{O}}(1)$ time) can be maintained with $\tilde{\mathcal{O}}(1)$ time per update and whether period queries for aperiodic substrings can be answered efficiently in a dynamic string.

References

- 1 Paniz Abedin, Sahar Hooshmand, Arnab Ganguly, and Sharma V. Thankachan. The Heaviest Induced Ancestors Problem Revisited. In *Annual Symposium on Combinatorial Pattern Matching, CPM 2018, July 2-4, 2018 - Qingdao, China*, pages 20:1–20:13, 2018. doi:10.4230/LIPIcs.CPM.2018.20.
- 2 Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New Data Structures for Orthogonal Range Searching. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 198–207. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892088.
- 3 Amihod Amir and Itai Boneh. Locally Maximal Common Factors as a Tool for Efficient Dynamic String Algorithms. In *Annual Symposium on Combinatorial Pattern Matching, CPM 2018, July 2-4, 2018 - Qingdao, China*, pages 11:1–11:13, 2018. doi:10.4230/LIPIcs.CPM.2018.11.
- 4 Amihod Amir and Itai Boneh. Dynamic Palindrome Detection. *CoRR*, abs/1906.09732, 2019. arXiv:1906.09732.
- 5 Amihod Amir, Panagiotis Charalampopoulos, Costas S. Iliopoulos, Solon P. Pissis, and Jakub Radoszewski. Longest Common Factor After One Edit Operation. In Gabriele Fici, Marinella Sciortino, and Rossano Venturini, editors, *String Processing and Information Retrieval: 24th International Symposium, SPIRE 2017, Proceedings*, volume 10508 of *Lecture Notes in Computer Science*, pages 14–26, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-67428-5_2.

- 6 Amihood Amir, Panagiotis Charalampopoulos, Solon P. Pissis, and Jakub Radoszewski. Longest Common Substring Made Fully Dynamic. In Ola Svensson Michael A. Bender and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, Munich/Garching, Germany, September 9-11, 2019*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019.
- 7 Amihood Amir and Martin Farach. Adaptive Dictionary Matching. In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 760–766. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185445.
- 8 Amihood Amir, Martin Farach, Ramana M. Idury, Johannes A. La Poutré, and Alejandro A. Schäffer. Improved Dynamic Dictionary Matching. *Inf. Comput.*, 119(2):258–282, 1995. doi:10.1006/inco.1995.1090.
- 9 Amihood Amir, Gad M. Landau, Moshe Lewenstein, and Dina Sokol. Dynamic text and static pattern matching. *ACM Trans. Algorithms*, 3(2):19, 2007. doi:10.1145/1240233.1240242.
- 10 Alberto Apostolico and Franco P. Preparata. Optimal Off-Line Detection of Repetitions in a String. *Theor. Comput. Sci.*, 22:297–315, 1983. doi:10.1016/0304-3975(83)90109-3.
- 11 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "Runs" Theorem. *SIAM J. Comput.*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.
- 12 Gary Benson. Tandem repeats finder: a program to analyze DNA sequences. *Nucleic Acids Research*, 27(2):573–580, January 1999. doi:10.1093/nar/27.2.573.
- 13 Philip Bille, Anders Roy Christiansen, Patrick Hage Cording, Inge Li Gørtz, Frederik Rye Skjoldjensen, Hjalte Wedel Vildhøj, and Søren Vind. Dynamic Relative Compression, Dynamic Partial Sums, and Substring Concatenation. *Algorithmica*, 80(11):3207–3224, 2018. doi:10.1007/s00453-017-0380-7.
- 14 Timothy M. Chan and Konstantinos Tsakalidis. Dynamic Orthogonal Range Searching on the RAM, Revisited. In Boris Aronov and Matthew J. Katz, editors, *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*, volume 77 of *LIPIcs*, pages 28:1–28:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.SoCG.2017.28.
- 15 Maxime Crochemore. An Optimal Algorithm for Computing the Repetitions in a Word. *Inf. Process. Lett.*, 12(5):244–250, 1981. doi:10.1016/0020-0190(81)90024-7.
- 16 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.
- 17 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Extracting powers and periods in a word from its runs structure. *Theor. Comput. Sci.*, 521:29–41, 2014. doi:10.1016/j.tcs.2013.11.018.
- 18 Camil Demetrescu, David Eppstein, Zvi Galil, and Giuseppe F. Italiano. Dynamic Graph Algorithms. In Mikhail J. Atallah and Marina Blanton, editors, *Algorithms and Theory of Computation Handbook: General Concepts and Techniques*, chapter 9. Chapman & Hall/CRC, 2010. URL: <http://dl.acm.org/citation.cfm?id=1882757.1882766>.
- 19 Nevzat Onur Domanic and Franco P. Preparata. A Novel Approach to the Detection of Genomic Approximate Tandem Repeats in the Levenshtein Metric. *Journal of Computational Biology*, 14(7):873–891, 2007. doi:10.1089/cmb.2007.0018.
- 20 Paolo Ferragina. Dynamic Text Indexing under String Updates. *J. Algorithms*, 22(2):296–328, 1997. doi:10.1006/jagm.1996.0814.
- 21 Paolo Ferragina and Fabrizio Luccio. Dynamic Dictionary Matching in External Memory. *Inf. Comput.*, 146(2):85–99, 1998. doi:10.1006/inco.1998.2733.
- 22 N. J. Fine and H. S. Wilf. Uniqueness Theorems for Periodic Functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. URL: <http://www.jstor.org/stable/2034009>.

- 23 Mitsuru Funakoshi, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Longest substring palindrome after edit. In *Annual Symposium on Combinatorial Pattern Matching, CPM 2018*, pages 12:1–12:14, 2018. doi:10.4230/LIPIcs.CPM.2018.12.
- 24 Mitsuru Funakoshi, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Faster Queries for Longest Substring Palindrome After Block Edit. In Nadia Pisanti and Solon P. Pissis, editors, *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy*, volume 128 of *LIPIcs*, pages 27:1–27:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.CPM.2019.27.
- 25 Pawel Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Lacki, and Piotr Sankowski. Optimal Dynamic Strings. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1509–1528. SIAM, 2018. doi:10.1137/1.9781611975031.99.
- 26 Ming Gu, Martin Farach, and Richard Beigel. An Efficient Algorithm for Dynamic Text Indexing. In Daniel Dominic Sleator, editor, *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia.*, pages 697–704. ACM/SIAM, 1994. URL: <http://dl.acm.org/citation.cfm?id=314464.314675>.
- 27 Dan Gusfield and Jens Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *J. Comput. Syst. Sci.*, 69(4):525–546, 2004.
- 28 Dan Gusfield and Jens Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *J. Comput. Syst. Sci.*, 69(4):525–546, 2004.
- 29 Ramana M. Idury and Alejandro A. Schäffer. Dynamic Dictionary Matching with Failure Functions. *Theor. Comput. Sci.*, 131(2):295–310, 1994. doi:10.1016/0304-3975(94)90176-7.
- 30 Richard M. Karp, Raymond E. Miller, and Arnold 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.
- 31 Richard M. Karp and Michael O. Rabin. Efficient Randomized Pattern-Matching Algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987. doi:10.1147/rd.312.0249.
- 32 Tomasz Kociumaka. *Efficient Data Structures for Internal Queries in Texts*. PhD thesis, University of Warsaw, 2018. URL: <https://mimuw.edu.pl/~kociumaka/files/phd.pdf>.
- 33 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Efficient Data Structures for the Factor Periodicity Problem. In Liliana Calderón-Benavides, Cristina N. González-Caro, Edgar Chávez, and Nivio Ziviani, editors, *String Processing and Information Retrieval - 19th International Symposium, SPIRE 2012, Cartagena de Indias, Colombia, October 21-25, 2012. Proceedings*, volume 7608 of *Lecture Notes in Computer Science*, pages 284–294. Springer, 2012. doi:10.1007/978-3-642-34109-0_30.
- 34 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal Pattern Matching Queries in a Text and Applications. 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 532–551. SIAM, 2015. doi:10.1137/1.9781611973730.36.
- 35 Roman M. Kolpakov and Gregory 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, 1999. doi:10.1109/SFCS.1999.814634.
- 36 Gad M. Landau, Jeanette P. Schmidt, and Dina Sokol. An Algorithm for Approximate Tandem Repeats. *Journal of Computational Biology*, 8(1):1–18, 2001. doi:10.1089/106652701300099038.
- 37 Michael G. Main and Richard J. Lorentz. An $O(n \log n)$ Algorithm for Finding All Repetitions in a String. *J. Algorithms*, 5(3):422–432, 1984. doi:10.1016/0196-6774(84)90021-X.
- 38 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.

- 39 Süleyman Cenk Sahinalp and Uzi Vishkin. Efficient Approximate and Dynamic Matching of Patterns Using a Labeling Paradigm (extended abstract). In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 320–328. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548491.
- 40 Dina Sokol, Gary Benson, and Justin Tojeira. Tandem repeats over the edit distance. *Bioinformatics*, 23(2):30–35, 2007. doi:10.1093/bioinformatics/bt1309.
- 41 Ydo Wexler, Zohar Yakhini, Yechezkel Kashi, and Dan Geiger. Finding Approximate Tandem Repeats in Genomic Sequences. *Journal of Computational Biology*, 12(7):928–942, 2005. doi:10.1089/cmb.2005.12.928.


Longest Common Substring Made Fully Dynamic

Amihood Amir

Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
amir@esc.biu.ac.il

Panagiotis Charalampopoulos 

Department of Informatics, King's College London, London, UK
Efi Arazi School of Computer Science, The Interdisciplinary Center Herzliya, Herzliya, Israel
panagiotis.charalampopoulos@kcl.ac.uk

Solon P. Pissis 

CWI, Amsterdam, The Netherlands
solon.pissis@cwi.nl

Jakub Radoszewski 

Institute of Informatics, University of Warsaw, Warsaw, Poland
Samsung R&D Institute, Warsaw, Poland
jrad@mimuw.edu.pl

Abstract

Given two strings S and T , each of length at most n , the longest common substring (LCS) problem is to find a longest substring common to S and T . This is a classical problem in computer science with an $\mathcal{O}(n)$ -time solution. In the fully dynamic setting, edit operations are allowed in either of the two strings, and the problem is to find an LCS after each edit. We present the first solution to this problem requiring sublinear time in n per edit operation. In particular, we show how to find an LCS after each edit operation in $\tilde{\mathcal{O}}(n^{2/3})$ time, after $\tilde{\mathcal{O}}(n)$ -time and space preprocessing.¹

This line of research has been recently initiated in a somewhat restricted dynamic variant by Amir et al. [SPIRE 2017]. More specifically, they presented an $\tilde{\mathcal{O}}(n)$ -sized data structure that returns an LCS of the two strings after a single edit operation (that is reverted afterwards) in $\tilde{\mathcal{O}}(1)$ time. At CPM 2018, three papers (Abedin et al., Funakoshi et al., and Urabe et al.) studied analogously restricted dynamic variants of problems on strings. We show that the techniques we develop can be applied to obtain fully dynamic algorithms for all of these variants. The only previously known sublinear-time dynamic algorithms for problems on strings were for maintaining a dynamic collection of strings for comparison queries and for pattern matching, with the most recent advances made by Gawrychowski et al. [SODA 2018] and by Clifford et al. [STACS 2018].

As an intermediate problem we consider computing the solution for a string with a given set of k edits, which leads us, in particular, to answering *internal* queries on a string. The input to such a query is specified by a substring (or substrings) of a given string. Data structures for answering internal string queries that were proposed by Kociumaka et al. [SODA 2015] and by Gagie et al. [CCCG 2013] are used, along with new ones, based on ingredients such as the suffix tree, heavy-path decomposition, orthogonal range queries, difference covers, and string periodicity.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases longest common substring, string algorithms, dynamic algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.6

Related Version A full version of the paper is available at <https://arxiv.org/abs/1804.08731>.

Funding *Amihood Amir*: Supported by Israel Science Foundation (ISF) grant 1475/18 and United States – Israel Binational Science Foundation (BSF) grant 2018141.

Panagiotis Charalampopoulos: Partially supported by Israel Science Foundation (ISF) grant 794/13.

Jakub Radoszewski: Supported by the “Algorithms for text processing with errors and uncertainties” project carried out within the HOMING programme of the Foundation for Polish Science co-financed by the European Union under the European Regional Development Fund.

¹ The $\tilde{\mathcal{O}}(\cdot)$ notation suppresses $\log^{\mathcal{O}(1)} n$ factors.



1 Introduction

Given two strings S and T , each of length at most n , the longest common substring (LCS) problem, also known as the longest common factor problem, is to find a longest substring common to S and T . This is a classical problem in theoretical computer science. Knuth had conjectured that the LCS problem was in $\Omega(n \log n)$. In 1973 Weiner solved it in the optimal $\mathcal{O}(n)$ time [49] designing a data structure that was later called the suffix tree (see also [21]). Knuth declared Weiner’s algorithm the “Algorithm of the Year” [11]. Since $\mathcal{O}(n)$ time is optimal for this problem, a series of studies have been dedicated in improving the working space [37, 44]. The LCS problem has also been studied under Hamming and edit distance. We refer the interested reader to [1, 43, 47, 16, 46, 12] and references therein.

In [43], Starikovskaya mentions that an answer to the LCS problem “is not robust and can vary significantly when the input strings are changed even by one character”, implicitly posing the following question: *Can we compute an LCS after editing S or T in $o(n)$ time?*

► **Example 1.** The length of an LCS of S and T below is *doubled* when substitution $S[4] := a$ is performed. The next substitution, $T[3] := b$, *halves* the length of an LCS.

$S = \underline{caabaaa}$	$S[4] := a$	$S = \underline{caaaaaa}$	$T[3] := b$	$S = \underline{caaaaaa}$
$T = \underline{aaaaaab}$		$T = \underline{aaaaaab}$		$T = \underline{aabaaab}$

This question poses the challenge of dynamically updating the suffix tree in the presence of edit operations (i.e. insertions, deletions and substitutions), which remains the main obstacle for answering this type of questions.

Amir et al. [8] introduced a restricted dynamic variant, where any *single* edit operation is allowed and is reverted afterwards. We call this problem LCS AFTER ONE EDIT. Amir et al. presented an $\tilde{\mathcal{O}}(n)$ -sized data structure that can be constructed in $\tilde{\mathcal{O}}(n)$ time supporting $\tilde{\mathcal{O}}(1)$ -time computation of an LCS, after one edit operation is applied on S . This work initiated a new line of research on analogously restricted dynamic variants of problems on strings [26, 27, 48]. Abedin et al. [3] improved the complexities of the data structure proposed by Amir et al. [8] by $\log^{\mathcal{O}(1)} n$ factors. Two other restricted variants of the dynamic LCS problem were considered by Amir and Boneh in [5]. In both variants substitutions were allowed in one of the strings; one was of decremental nature and in the other one the complexity was parameterized by the period of the static string.

In this paper we make substantial progress: we show a strongly sublinear-time solution for the general version of the problem, namely, the fully dynamic case of the LCS problem. Given two strings S and T , the problem is to answer the following type of queries in an on-line manner: perform an edit operation (substitution, insertion, or deletion) on S or on T and then return an LCS of the new S and T . We call this problem FULLY DYNAMIC LCS.

Below we mention some of the known results on dynamic problems on strings.

Dynamic Pattern Matching. Finding all *occ* occurrences of a pattern of length m in a *static* text can be performed in the optimal $\mathcal{O}(m + occ)$ time using suffix trees, which can be constructed in linear time [49, 21]. In the fully dynamic setting, the problem is to compute the new set of occurrences when allowing for edit operations anywhere on the text. A considerable amount of work has been carried out on this problem [31, 22, 23]. The first data structure with polylogarithmic update time and time-optimal queries was shown by Sahinalp and Vishkin [41]. The update time was later improved by Alstrup et al. [4] at the expense of slightly suboptimal query time. The state of the art is the data structure by

Gawrychowski et al. [29] supporting time-optimal queries with $\mathcal{O}(\log^2 n)$ time for updates. Clifford et al. [18] have recently shown upper and lower bounds for variants of exact matching with wildcard characters, inner product, and Hamming distance.

Dynamic String Collection with Comparison. The problem is to maintain a dynamic collection \mathcal{W} of strings of total length n supporting the following operations: adding a string to \mathcal{W} , adding the concatenation of two strings from \mathcal{W} to \mathcal{W} , splitting a string from \mathcal{W} and adding the two residual strings in \mathcal{W} , and returning the length of the longest common prefix of two strings from \mathcal{W} . This line of research was initiated by Sundar and Tarjan [45]. Data structures supporting updates in polylogarithmic time were presented by Mehlhorn et al. [40] and Alstrup et al. [4]. Finally, Gawrychowski et al. [30] proposed an optimal solution.

Longest Palindrome and Longest Lyndon Substring. A string is called *palindrome* if it is the same as its reverse. A string is called *Lyndon* if it is smaller lexicographically than all its suffixes [38]. Computing a longest palindrome and a longest Lyndon substring of a string after a single edit have been recently studied in [26] (see also [27]) and in [48], respectively.

Maintaining Repetitions. *Squares* are strings of the form XX . In [7], the authors show how to maintain squares in a dynamic string S of length n in $n^{o(1)}$ time per operation. A modification of this algorithm, with the same time complexity per operation, allows them to determine in $\tilde{\mathcal{O}}(1)$ time whether a queried substring of S is periodic, and if so, compute its period.

Our Results. We give the first fully dynamic algorithm for the LCS problem that works in *strongly sublinear* time per edit operation in any of the two strings. Specifically, for two strings, each of length up to n , it computes an LCS after each edit operation in $\tilde{\mathcal{O}}(n^{2/3})$ time after $\tilde{\mathcal{O}}(n)$ -time and space preprocessing. To ease the comprehension of the algorithm for FULLY DYNAMIC LCS, we first show a solution of an auxiliary problem called LCS AFTER ONE SUBSTITUTION PER STRING, where a single substitution is allowed in each of the strings and is reverted afterwards, with $\tilde{\mathcal{O}}(1)$ -time queries after $\tilde{\mathcal{O}}(n)$ -time and space preprocessing.

Notably, we showcase the applicability of our techniques to other string problems in the fully dynamic setting. We present a fully dynamic algorithm for computing a longest repeat of a string S of length n , i.e. a longest substring occurring more than once in S , in $\tilde{\mathcal{O}}(n^{2/3})$ time. We also present a fully dynamic algorithm for computing a longest palindrome substring of a string S requiring $\tilde{\mathcal{O}}(\sqrt{n})$ time per edit. Finally, we present a fully dynamic algorithm, requiring $\tilde{\mathcal{O}}(\sqrt{n})$ time per edit, for computing a longest Lyndon substring of string S as well as maintaining a representation of the Lyndon factorization of S that allows us to efficiently extract the t -th element of the factorization in $\tilde{\mathcal{O}}(1)$ time.

Our data structure is randomized due to the use of data structures for dynamic strings [30] and internal pattern matching [35]; the latter can be derandomized [34].

Roadmap. Section 2 provides the necessary definitions and notation used throughout as well as the standard algorithmic toolbox for string processing and the general scheme of our approach. In Section 3 we show an optimal, up to polylogarithmic factors, solution for LCS AFTER ONE SUBSTITUTION PER STRING. In Section 4 we show our main result: a solution for FULLY DYNAMIC LCS. Some technical details, including details on several special cases of internal LCS queries, are omitted in this version. A brief overview of our fully dynamic algorithms for computing the longest repeat, the longest palindrome, and the longest Lyndon substring of a string is provided in Section 5. We conclude this work in Section 6.

2 Preliminaries

Strings. Let $S = S[1]S[2] \dots S[n]$ be a *string* of length $|S| = n$ over an integer alphabet $\Sigma = \{1, \dots, n^{\mathcal{O}(1)}\}$. The elements of Σ are called *characters*. For two positions i and j on S , we denote by $S[i..j] = S[i] \dots S[j]$ the substring of S that starts at position i and ends at position j (it is empty if $i > j$). A substring of S is represented in $\mathcal{O}(1)$ space by specifying the indices i and j . A prefix $S[1..j]$ is denoted by $S^{(j)}$ and a suffix $S[i..n]$ is denoted by $S_{(i)}$. A substring of S is called *proper* if it is shorter than S . We denote the *reverse string* of S by $S^R = S[n]S[n-1] \dots S[1]$. By ST , S^k , and S^∞ we denote the concatenation of strings S and T , k copies of string S , and infinitely many copies of string S , respectively. If a string B is both a proper prefix and a proper suffix of string S , then B is called a *border* of S . A positive integer p is called a *period* of S if $S[i] = S[i+p]$ for all $i = 1, \dots, n-p$. String S has a period p if and only if it has a border of length $n-p$. We refer to the smallest period as *the period* of the string and, analogously, to the longest border as *the border* of the string.

The *suffix tree* $\mathcal{T}(S)$ of string S is a compact trie representing all suffixes of S . The suffix tree of a string of length n over an integer alphabet can be constructed in $\mathcal{O}(n)$ time and space [21]. By $\text{lcpstring}(S, T)$ we denote the longest common prefix of S and T , by $\text{lcp}(S, T)$ we denote $|\text{lcpstring}(S, T)|$, and by $\text{lcp}(r, s)$ we denote $\text{lcp}(S_{(r)}, S_{(s)})$. Further by $\text{lcssstring}(S, T)$ we denote the longest common suffix of S and T . An $\mathcal{O}(n)$ -sized lowest common ancestor data structure can be constructed over the suffix tree of S in $\mathcal{O}(n)$ time [14], supporting $\text{lcp}(r, s)$ -queries in $\mathcal{O}(1)$ time. A symmetric construction on S^R (the reverse of S) can answer the so-called *longest common suffix* (lcs) queries in the same complexity. The lcp and lcs queries are also known as *longest common extension* (LCE) queries.

General Scheme and Relation to Internal Pattern Matching. The scheme of our approach for most of the considered dynamic problems on strings is as follows. Let the input be a string S of length n (in the case of the LCS problem, this can be the concatenation of the input strings S and T separated by a delimiter). We construct a data structure that answers the following type of queries: given k edit operations on S , compute the answer to a particular problem on the resulting string S' . Assuming that the data structure occupies $\mathcal{O}(s_n)$ space, answers queries for k edits in time $\mathcal{O}(q_n(k))$ and can be constructed in time $\mathcal{O}(t_n)$ ($s_n \geq n$ and $q_n(k) \geq k$ is non-decreasing with respect to k), this data structure can be used to design a dynamic algorithm that preprocesses the input string in time $\mathcal{O}(t_n)$ and answers queries dynamically under edit operations in amortized time $\mathcal{O}(q_n(\kappa))$, where κ is such that $q_n(\kappa) = (t_n + n)/\kappa$, using $\mathcal{O}(s_n)$ space. The query time can be made worst-case using *time slicing*: for $s_n, t_n = \tilde{\mathcal{O}}(n)$ and $q_n(k) = \tilde{\mathcal{O}}(k)$ we obtain a fully dynamic algorithm with $\tilde{\mathcal{O}}(\sqrt{n})$ -time queries, whereas for $q_n(k) = \tilde{\mathcal{O}}(k^2)$ the query time is $\tilde{\mathcal{O}}(n^{2/3})$.

A *k-substring* of a string S is a concatenation of k strings, each of which is either a substring of S or a single character. A k -substring of S can be represented in $\mathcal{O}(k)$ additional space using a doubly-linked list if the string S itself is stored. The string S after k subsequent edit operations can be represented as a $(2k+1)$ -substring due to the following lemma.

► **Lemma 2.** *Let S' be a k -substring of S and S'' be S' after a single edit operation. Then S'' is a $(k+2)$ -substring of S . Moreover, S'' can be computed from S' in $\mathcal{O}(k)$ time.*

Proof. Let $S' = F_1 \dots F_k$ where each F_i is either a substring of S or a single character. We traverse the list of substrings until we find the substring F_i such that the edit operation takes place at the j -th character of F_i . As a result, F_i is decomposed into a prefix and a suffix, potentially with a single character inserted in between in case of insertion or substitution. The resulting string S'' is a $(k+2)$ -substring of S . ◀

Thus the fully dynamic version reduces to designing a data structure over a string S of length n that computes the result of a specific problem on a k -substring $F_1 \dots F_k$ of S . For the considered problems we aim at computing the longest substring of S that satisfies a certain property. Then there are two cases. Case 1: the sought substring occurs inside one of the substrings F_i (or each of its two occurrences satisfies this property in case of the LCS and the longest repeat problems). Case 2: it contains the boundary between some two substrings F_i and F_{i+1} . Case 1 requires to compute the solution to a certain problem on a substring or substrings of a specified string. This is the so-called *internal* model of queries; this name was coined by Kociumaka et al. in [35]. We call Case 2 *cross-substring* queries. Due to string periodicity, certain internal queries arise in cross-substring queries as well.

3 LCS After One Substitution Per String

Let us now consider an extended version of the LCS AFTER ONE EDIT problem, for simplicity restricted to substitutions.

LCS AFTER ONE SUBSTITUTION PER STRING

Input: Two strings S and T of length at most n

Query: For given indices i, j and characters α and β , compute $\text{LCS}(S', T')$ where S' is S after substitution $S[i] := \alpha$ and T' is T after substitution $T[j] := \beta$

To solve this problem we consider three cases depending on whether an occurrence of the LCS contains any of the changed positions in S and T . We prove the following result.

► **Theorem 3.** LCS AFTER ONE SUBSTITUTION PER STRING can be computed in $\tilde{O}(1)$ time after $\tilde{O}(n)$ -time and space preprocessing.

3.1 LCS Contains No Changed Position

We use the following lemma for a special case of internal LCS queries. Its proof is deferred to the full version. In the fully dynamic algorithm a less restrictive approach is necessary.

► **Lemma 4.** Let S and T be two strings of length at most n . After $\mathcal{O}(n \log^2 n)$ -time and $\mathcal{O}(n \log n)$ -space preprocessing, an LCS between any prefix or suffix of S and prefix or suffix of T can be computed in $\mathcal{O}(\log n)$ time.

It suffices to apply internal LCS queries of Lemma 4 four times: each time for one of $S^{(i-1)}, S_{(i+1)}$ and one of $T^{(j-1)}, T_{(j+1)}$.

3.2 LCS Contains a Changed Position in Exactly One of the Strings

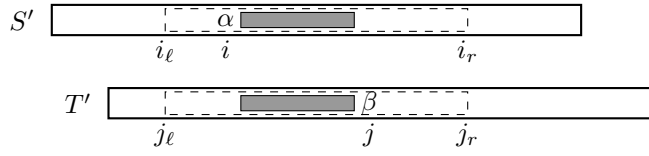
We use the following lemma that encapsulates one of the main techniques of [8]. It involves computing so-called *ranges* of substrings in the *generalized suffix array* of S and T and it relies on a result by Fischer et al. [24].

► **Lemma 5.** Let S and T be strings of length at most n . After $\mathcal{O}(n \log \log n)$ -time and $\mathcal{O}(n)$ -space preprocessing, given two substrings P and Q of S or T , we can compute:

- (a) a substring of T equal to PQ , if it exists, in $\mathcal{O}(\log \log n)$ time;
- (b) the longest substring of T that is a prefix (or a suffix) of PQ in $\mathcal{O}(\log n \log \log n)$ time.

We now show how to compute the longest substring that contains the position i in S , but not the position j in T (the opposite case is symmetric). We first use Lemma 5(b) to compute two substrings, U and V , of T in $\mathcal{O}(\log n \log \log n)$ time:

- U is the longest substring of T that is equal to a suffix of $S[1 \dots i - 1]$;
- V is the longest substring of T that is equal to a prefix of $\alpha S[i + 1 \dots |S|]$.



■ **Figure 1** Occurrences of an LCS of S' and T' containing both changed positions are denoted by dashed rectangles. Occurrences of U at which an LCS is aligned are denoted by gray rectangles.

Our task then reduces to computing the longest substring of UV that crosses the boundary between U and V and is a substring of $T^{(j-1)}$ or of $T_{(j+1)}$. We can compute it using the following type of queries.

THREE SUBSTRINGS LCS
Input: A string T
Query: Given three substrings U , V , and W of T , compute the longest substring XY of W such that X is a suffix of U and Y is a prefix of V

Indeed, it suffices to ask two THREE SUBSTRINGS LCS queries: one with $W = T^{(j-1)}$ and one with $W = T_{(j+1)}$.

A solution to a special case of THREE SUBSTRINGS LCS queries with $W = T$ was already implicitly presented by Amir et al. in [8]. It is based on the *heaviest induced ancestors* (HIA) problem on trees, introduced by Gagie et al. [28], applied to the suffix tree of T . We generalize the HIA queries and use them to answer general THREE SUBSTRINGS LCS queries. The data structure for answering our generalization of HIA queries turns out to be one of the most technical parts of the paper. It relies on the construction of multidimensional grids for pairs of heavy paths (in heavy-path decompositions [42]) of the involved trees. Each query can be answered by interpreting the answer of $\mathcal{O}(\log^2 n)$ orthogonal range maximum queries over such grids.

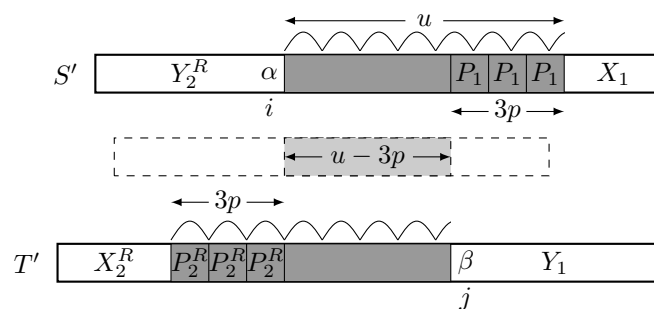
► **Lemma 6.** *Let T be a string of length at most n . After $\tilde{\mathcal{O}}(n)$ -time preprocessing, we can answer THREE SUBSTRINGS LCS queries in $\tilde{\mathcal{O}}(1)$ time.*

3.3 LCS Contains a Changed Position in Each of the Strings

A Prefix-Suffix Query gets as input two substrings X and Y of a string S of length n and an integer d and returns the lengths of all prefixes of X of length between d and $2d$ that are suffixes of Y . It is known that such a query returns an arithmetic sequence and if it has at least three elements, then its difference equals the period of all the corresponding prefixes-suffixes. Moreover, Kociumaka et al. [35] show that Prefix-Suffix Queries can be answered in $\mathcal{O}(1)$ time using a data structure of $\mathcal{O}(n)$ size, which can be constructed in $\mathcal{O}(n)$ time. By considering $X = Y = U$, this implies the two respective points of the lemma below.

► **Lemma 7.**

- (a) *For a string U of length m , the set $\mathcal{B}_r(U)$ of border lengths of U between 2^r and $2^{r+1} - 1$ is an arithmetic sequence. If it has at least three elements, all the corresponding borders have the same period, equal to the difference of the sequence.*
- (b) *[35] Let S be a string of length n . For any substring U of S and integer r , the arithmetic sequence $\mathcal{B}_r(U)$ can be computed in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$ -time and space preprocessing.*



■ **Figure 2** A border of length u is denoted by dark gray rectangles. An LCS aligned at a border of length $u - 3p$, which is in the same arithmetic sequence, is denoted by the dashed rectangle.

We next show an algorithm that finds a longest string $S'[i_\ell \dots i_r] = T'[j_\ell \dots j_r]$ such that $i_\ell \leq i \leq i_r$ and $j_\ell \leq j \leq j_r$ for the given indices i, j . Let us assume that $i - i_\ell \leq j - j_\ell$; the symmetric case can be treated analogously. We have that $U \stackrel{\text{def}}{=} S'[i + 1 \dots i_\ell + j - j_\ell - 1] = T'[j_\ell + i - i_\ell + 1 \dots j - 1]$ as shown in Figure 1. ($U = \varepsilon$ can correspond to $i - i_\ell = j - j_\ell$ or $i - i_\ell + 1 = j - j_\ell$, so both these cases need to be checked.) Note that these substrings do not contain any changed position. Any such U is a prefix of $S_{(i+1)}$ and a suffix of $T^{(j-1)}$; let U_0 denote the longest such string. Then, the possible candidates for U are U_0 and all its borders. For a border U of U_0 , we say that $\text{lcsstring}(S'^{(i)}, T'^{(j-|U|-1)}) U \text{lcpstring}(S'_{(i+|U|+1)}, T'_{(j)})$ is an *LCS aligned at U* . We compute U_0 in time $\mathcal{O}(\log n)$ by asking Prefix-Suffix Queries for $X = S_{(i+1)}$, $Y = T^{(j-1)}$ in $S\#T$ and $d = 2^r$ for all $r = 0, 1, \dots, \lfloor \log j \rfloor$. We then consider the borders of U_0 in arithmetic sequences of their lengths; see Lemma 7. If an arithmetic sequence has at most two elements, we compute an LCS aligned at each of the borders in $\mathcal{O}(1)$ time by the above formula using LCE queries. Otherwise, let p be the difference of the arithmetic sequence, ℓ be its length, and u be its maximum element. Further let:

$$\begin{aligned} X_1 &= S'_{(i+u+1)}, & Y_1 &= T'_{(j)}, & P_1 &= S'[i + u - p + 1 \dots i + u], \\ X_2^R &= T'^{(j-u-1)}, & Y_2^R &= S'^{(i)}, & P_2^R &= T'[j - u \dots j - u + p - 1]. \end{aligned}$$

The setting is presented in Figure 2. It can be readily verified (inspect Figure 2) that a longest common substring aligned at the border of length $u - wp$, for $w \in [0, \ell - 1]$, is equal to

$$\text{lcs}(X_2^R (P_2^R)^w, Y_2^R) + u - wp + \text{lcp}(P_1^w X_1, Y_1) = \text{lcp}(P_2^w X_2, Y_2) + \text{lcp}(P_1^w X_1, Y_1) + u - wp$$

which we further denote by $g(w)$. Thus, a longest LCS aligned at a border whose length is in this arithmetic sequence is $\max_{w=0}^{\ell-1} g(w)$. The following observation facilitates efficient evaluation of this formula.

► **Observation 8.** *For any strings P, X, Y , the function $f(w) = \text{lcp}(P^w X, Y)$ for integer $w \geq 0$ is piecewise linear with at most three pieces. Moreover, if P, X, Y are substrings of a string S , then the exact formula of f can be computed with $\mathcal{O}(1)$ LCE queries on S .*

Proof. Let $a = \text{lcp}(P^\infty, X)$, $b = \text{lcp}(P^\infty, Y)$, and $p = |P|$. Then:

$$f(w) = \begin{cases} a + wp & \text{if } a + wp < b \\ w + \text{lcp}(X, Y[aw + 1 \dots |Y|]) & \text{if } a + wp = b \\ b & \text{if } a + wp > b. \end{cases}$$

Note that a can be computed from $\text{lcp}(P, X)$ and $\text{lcp}(X, X[p+1 \dots |X|])$, and b analogously. Thus if P, X, Y are substrings of S , five LCE queries on S suffice. ◀

By Observation 8, $g(w)$ can be expressed as a piecewise linear function with $\mathcal{O}(1)$ pieces. Moreover, its exact formula can be computed using $\mathcal{O}(1)$ LCE queries on $S' \# T'$, hence, in $\mathcal{O}(1)$ time using LCE queries. This allows to compute $\max_{w=0}^{\ell-1} g(w)$ in $\mathcal{O}(1)$ time. Each arithmetic sequence is processed in $\mathcal{O}(1)$ time. The global maximum that contains both changed positions is the required answer. Thus the query time in this case is $\mathcal{O}(\log n)$ and the preprocessing requires $\mathcal{O}(n)$ time and space.

By combining the results of Sections 3.1 to 3.3, we arrive at Theorem 3.

4 Fully Dynamic LCS

In this section we assume that the sought LCS has length at least 2. The case that it is of unit or zero length can be easily treated separately. We use the following auxiliary problem that generalizes LCS AFTER ONE SUBSTITUTION PER STRING into the case of k edit operations:

(k_1, k_2) -SUBSTRING LCS

Input: Two strings S and T of length at most n

Query: Compute $\text{LCS}(S', T')$ where $S' = F_1 \dots F_{k_1}$ is a k_1 -substring of S , $T' = G_1 \dots G_{k_2}$ is a k_2 -substring of T , and $k_1 + k_2 = k$

As in Section 3, we consider three cases listed below. The main difference in the approach takes place in the first case since the most general internal LCS queries are probably hard to answer. Indeed, this query can be reduced via a binary search to $\mathcal{O}(\log n)$ two-range-LCP queries of Amir et al. [10]. With their Theorem 6, we can construct a data structure of size $\mathcal{O}(n)$ in $\mathcal{O}(n\sqrt{n})$ time that allows for $\tilde{\mathcal{O}}(\sqrt{n})$ -time queries. We cannot use this data structure in our scheme though due to its high preprocessing cost. In fact, Amir et al. [10] show that the two-range-LCP data structure problem is at least as hard as the *Set Emptiness* problem: preprocess a collection of sets of total cardinality n so that queries of whether the intersection of two sets is empty can be answered efficiently. The best known $\mathcal{O}(n)$ -sized data structure for this problem has $\mathcal{O}(\sqrt{n/w})$ -query-time, where w is the size of the computer word. The reduction of [10] can be adapted to show that answering general internal LCS queries is at least as hard as answering Set Emptiness queries. In light of this, in the first case, we develop a different global approach to circumvent answering such queries.

1. An LCS does not contain any position (or boundary between positions) in S or T where an edit took place. As it was mentioned before, this problem probably cannot be solved efficiently in the language of k -substrings. Instead, we compute such an LCS via an inherently dynamic algorithm for the DECREMENTAL LCS problem. See Section 4.1.
2. An LCS contains at least one position where an edit operation took place in exactly one of the strings. This corresponds to the (k_1, k_2) -SUBSTRING LCS problem when an LCS contains the boundary between some substrings of exactly one of S' and T' . We compute such an LCS by combining the techniques of Section 3.2 with a sliding window approach. See Section 4.2.
3. An LCS contains at least one position where an edit operation took place in both of the strings. This corresponds to the (k_1, k_2) -SUBSTRING LCS problem when an LCS contains the boundary between some substrings in both of S' and T' . We compute such an LCS by building upon the techniques of Section 3.3 and employing efficient LCE queries for k -substrings. See Section 4.3.

4.1 Decremental LCS

We use the following convenient formulation of the problem, where the only letter that can be inserted or substituted in S (resp. in T) is $\# \notin \Sigma$, (resp. $\$ \notin \Sigma$), with $\# \neq \$$. An insertion in S in FULLY DYNAMIC LCS corresponds to an insertion of a $\#$, while both deletions and substitutions correspond to substitutions with a $\#$. T is treated similarly. We call the problem of reporting an LCS after each such operation DECREMENTAL LCS.

We first consider the case where the sought LCS (in the fully dynamic case) is of length bounded by d ; we call this problem d -BOUNDED-LENGTH LCS.

Before we proceed to describe a solution to this problem we discuss how to answer LCE queries efficiently in a dynamic string. We resort to the main result of Gawrychowski et al. [30] to obtain the following lemma.

► **Lemma 9.** *A string S of length n can be preprocessed in $\mathcal{O}(n)$ time and space so that $k = \mathcal{O}(n)$ edit operations and $m = \mathcal{O}(n)$ lcp queries, in any order, can be processed in $\mathcal{O}(\log n)$ time each, using $\mathcal{O}(k \log n + m \log n)$ space in total.*

► **Lemma 10.** *d -BOUNDED-LENGTH LCS can be solved in $\mathcal{O}(d \log^2 n)$ time per operation after $\tilde{\mathcal{O}}(n)$ -time preprocessing, using $\tilde{\mathcal{O}}(n + kd)$ space for k performed operations.*

Proof. Let U and V be the multisets of d -length substrings and the $d - 1$ suffixes of length smaller than d of S and T , respectively. We will maintain balanced BSTs B_X , with respect to the lexicographical order, containing the elements of X , for $X = U, V$, stored as substrings. We can search in these balanced BSTs in $\mathcal{O}(\log^2 n)$ since a comparison in it is an lcp query, which requires $\mathcal{O}(\log n)$ by Lemma 9, possibly followed by a character comparison. Each node of B_X will maintain a counter denoting its multiplicity in X . Let $Y = U \cup V$; we do not use Y in the algorithm, we just introduce it for conceptual convenience.

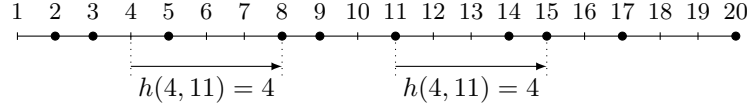
► **Observation 11.** *The length of the LCS of length at most d is equal to the maximum lcp between pairs of consecutive substrings in (the sorted) Y that originate from different strings.*

During preprocessing, we compute the lcp of all pairs described in Observation 11 and store them in a max heap H . To each element of the heap, we store a pointer from the nodes $u \in B_U, v \in B_V$ it originates from.

Each edit in S or T yields $\mathcal{O}(d)$ deletions and $\mathcal{O}(d)$ insertions of substrings in each of U, V and Y . We first perform deletions and then insertions. For each such operation, we have to check if it destroys or creates a pair of consecutive elements in (the sorted) Y , originating from different strings. We observe that upon the insertion/deletion of a string P , only pairs involving $P, \text{pred}_U(P), \text{pred}_V(P), \text{succ}_U(P)$ and $\text{succ}_V(P)$ may be involved, where pred, succ are predecessor and successor with respect to the lexicographical order. These elements can be identified in $\mathcal{O}(\log^2 n)$ time. The max heap can then be updated using a constant number of LCE queries and heap updates. By Lemma 9, LCE queries (and heap updates) require $\mathcal{O}(\log n)$ time each. Finally, we return the maximum element of the heap. ◀

We now focus on the harder case that the sought LCS is of length at least d .

Let S' and T' be the strings S and T after p operations; for some $p \leq k$. For a position i , by $\text{succ}_{S'}^\#(i)$ we denote the smallest position $j \geq i$ such that $S'[j] = \#$. If no such position exists, we set $\text{succ}_{S'}^\#(i) = |S'| + 1$. Similarly, by $\text{pred}_{S'}^\#(i)$ we denote the greatest position $j \leq i$ such that $S'[j] = \#$, or 0 if no such position exists. Similarly we define $\text{succ}_{T'}^\$(i)$ and $\text{pred}_{T'}^\$(i)$. Such values can be computed in $\mathcal{O}(\log n)$ time if the set of replaced positions is stored in a balanced BST (note that positions of $\#$ and $\$$ can be shifted due to insertions).



■ **Figure 3** An example of a 6-cover $\mathbf{S}_{20}(6) = \{2, 3, 5, 8, 9, 11, 14, 15, 17, 20\}$, with the elements marked as black circles. For example, we may have $h(4, 11) = 4$ since $4 + 4, 11 + 4 \in \mathbf{S}_{20}(6)$.

We say that a set $\mathbf{S}(d) \subseteq \mathbb{Z}_+$ is a d -cover if there is a constant-time computable function h such that for $i, j \in \mathbb{Z}_+$ we have $0 \leq h(i, j) < d$ and $i + h(i, j), j + h(i, j) \in \mathbf{S}(d)$.

► **Lemma 12** ([39, 15]). *For each $d \in \mathbb{Z}_+$ there is a d -cover $\mathbf{S}(d)$ such that $\mathbf{S}(d) \cap [1, n]$ is of size $\mathcal{O}(\frac{n}{\sqrt{d}})$ and can be constructed in $\mathcal{O}(\frac{n}{\sqrt{d}})$ time.*

The intuition behind applying the d -cover in our string-processing setting is as follows (inspect also Figure 3). Consider a position i on S and a position j on T . Note that $i, j \in [1, n]$. By the d -cover construction, we have that $h(i, j)$ is within distance d and $i + h(i, j), j + h(i, j) \in \mathbf{S}(d)$. Thus if we want to find a longest common substring of length *at least* d , it suffices to compute longest common extensions to the left and to the right of *only* positions $i', j' \in \mathbf{S}(d)$ (black circles in Figure 3) and then merge these partial results accordingly.

For this we use the following auxiliary problem that was introduced in [16].

TWO STRING FAMILIES LCP

Input: A compact trie $\mathcal{T}(\mathcal{F})$ of a family of strings \mathcal{F} and two sets $\mathcal{P}, \mathcal{Q} \subseteq \mathcal{F}^2$

Output: The value $\text{maxPairLCP}(\mathcal{P}, \mathcal{Q})$, defined as

$\text{maxPairLCP}(\mathcal{P}, \mathcal{Q}) = \max\{\text{lcp}(P_1, Q_1) + \text{lcp}(P_2, Q_2) : (P_1, P_2) \in \mathcal{P} \text{ and } (Q_1, Q_2) \in \mathcal{Q}\}$

An efficient solution to this problem was shown in [16] (and, implicitly, in [20, 25]).

► **Lemma 13** ([16]). *TWO STRING FAMILIES LCP can be solved in $\mathcal{O}(|\mathcal{F}| + N \log N)$ time, where $N = |\mathcal{P}| + |\mathcal{Q}|$.*

► **Lemma 14.** *DECREMENTAL LCS can be solved in $\tilde{\mathcal{O}}(n^{2/3})$ time per query, using $\tilde{\mathcal{O}}(n + kn^{2/3})$ space, after $\tilde{\mathcal{O}}(n)$ -time preprocessing for k performed operations.*

Proof. Let us consider an integer $d \in [1, n]$. For lengths up to d , we use the algorithm for the BOUNDED-LENGTH LCS problem of Lemma 10. If this problem indicates that there is a solution of length at least d , we proceed to the second step. Let $A = \mathbf{S}(d) \cap [1, n]$ be a d -cover of size $\mathcal{O}(n/\sqrt{d})$ (see Lemma 12).

We consider the following families of pairs of strings: $\mathcal{P} = \{(S[\text{pred}_S^\#(i-1) + 1 \dots i - 1])^R, S[i \dots \text{succ}_S^\#(i) - 1]) : i \in A\}$ and $\mathcal{Q} = \{(T[\text{pred}_T^\$(i-1) + 1 \dots i - 1])^R, T[i \dots \text{succ}_T^\$(i) - 1]) : i \in A\}$. We define \mathcal{F} as the family of strings that occur in the pairs from \mathcal{P} and \mathcal{Q} . Then $\text{maxPairLCP}(\mathcal{P}, \mathcal{Q})$ equals the length of the sought LCS, provided that it is at least d .

Note that $|\mathcal{P}|, |\mathcal{Q}|, |\mathcal{F}|$ are $\mathcal{O}(n/\sqrt{d})$. A compact trie $\mathcal{T}(\mathcal{F})$ can be constructed in $\mathcal{O}(|\mathcal{F}| \log |\mathcal{F}|)$ time by sorting all the strings (using lcp-queries) and then a standard left-to-right construction; see [19]. Thus we can use the solution to TWO STRING FAMILIES LCP which takes $\tilde{\mathcal{O}}(n/\sqrt{d})$ time. We set $d = \lfloor n^{2/3} \rfloor$ to obtain the stated complexity. ◀

4.2 One-Sided Cross-Substring Queries

We show a solution with $\tilde{\mathcal{O}}(k^2)$ -time queries after $\tilde{\mathcal{O}}(n)$ -time preprocessing by building upon the techniques from Section 3.2.

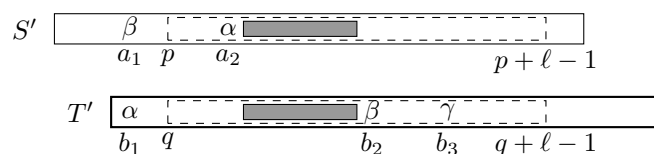


Figure 4 Occurrences of an LCS of S' and T' crossing the boundaries in both are denoted by dashed rectangles. The starting positions f_{i+1} and g_{j+1} minimize the formula $|(f_{i+1} - p) - (g_{j+1} - q)|$. Hence the gray rectangle, denoting U , is a prefix of $S'[f_{i+1} \dots f_{i+2} - 1]$ and a suffix of $T'[g_j \dots g_{j+1} - 1]$. We thus process it as a border while processing (F_i, F_{i+1}) and (G_j, G_{j+1}) and hence find this LCS.

We present an algorithm that computes, for each $i = 1, \dots, k_1$, the longest substring of S' that contains the first character of F_i , but not of F_{i-1} and occurs in G_p for a given $p \in \{1, \dots, k_2\}$ in $\tilde{\mathcal{O}}(k)$ time. These are the possible LCSs that cross the substring boundaries.

Let us start by a global part of the computation. For convenience let us assume that $F_0 = F_{k_1+1}$ are empty strings. For an index $i \in \{1, \dots, k_1\}$, by $next(i)$ we denote the greatest index $j \geq i - 1$ for which $F_i \dots F_j$ is a substring of T . These values are computed using a sliding-window-based approach. We start with computing $next(1)$. To this end, we use Lemma 5(a) for subsequent substrings F_1, F_2, \dots as long as their concatenation is a substring of T . This takes $\mathcal{O}(k \log \log n)$ time. Now assume that we have computed $next(i)$ and we wish to compute $next(i + 1)$. Obviously, $next(i + 1) \geq next(i)$. Let $j = next(i)$. We start with $F_{i+1} \dots F_j$ which is represented as a substring of T . We keep extending this substring by F_{j+1}, F_{j+2}, \dots using Lemma 5(a) as before as long as the concatenation is a substring of T . In total, computing values $next(i)$ for all $i = 1, \dots, k_1$ takes $\tilde{\mathcal{O}}(k)$ time.

Let us now fix i and let $j = next(i)$. We use Lemma 5(b) to find the longest prefix P_i of $(F_i \dots F_j)F_{j+1}$ that occurs in T ; it is also the longest prefix of $F_i \dots F_{k_1}$ that occurs in T by the definition of $next(i)$. Then Lemma 5(b) can be used to compute the longest suffix Q_i of F_{i-1} that occurs in T . For each i it takes time $\tilde{\mathcal{O}}(1)$ time to find P_i and Q_i after $\tilde{\mathcal{O}}(n)$ time and space preprocessing.

We then compute the sought result for given $i \in \{1, \dots, k_1\}$ and $p \in \{1, \dots, k_2\}$ by a THREE SUBSTRINGS LCS query for $U = Q_i$, $V = P_i$, and $W = G_p$. With Lemma 6 this takes $\tilde{\mathcal{O}}(k^2)$ time in total after $\tilde{\mathcal{O}}(n)$ time and space preprocessing.

4.3 Two-Sided Cross-Substring Queries

We show a solution with $\mathcal{O}(k^2 \log^3 n)$ -time queries after $\mathcal{O}(n \log n)$ -time preprocessing by combining the ideas presented in Section 3.3 and efficient LCE queries in the dynamic setting (cf. Lemma 9). We consider each pair of boundaries between pairs (F_i, F_{i+1}) and (G_j, G_{j+1}) , for $1 \leq i \leq k_1 - 1$ and $1 \leq j \leq k_2 - 1$. We process the prefixes of F_{i+1} that are suffixes of G_j as in Section 3.3 (the symmetric case is treated analogously).

We next argue that we do not miss any possible LCS by only considering such prefix-suffix pairs of F_{i+1} and G_j . Let f_i and g_i be the starting positions of F_i and G_j in S' and T' , respectively. An LCS $S'[p \dots p + \ell - 1] = T'[q \dots q + \ell - 1]$ of this type will be reported when processing the pairs (F_i, F_{i+1}) and (G_j, G_{j+1}) , satisfying $p \leq f_{i+1} \leq p + \ell - 1$, $q \leq g_{j+1} \leq q + \ell - 1$, for which $|(f_{i+1} - p) - (g_{j+1} - q)|$ is minimal. Without loss of generality assume $f_{i+1} - p \leq g_{j+1} - q$. Then, $S'[f_{i+1} \dots p + g_{j+1} - q - 1]$ is a prefix of F_{i+1} and $T'[q + f_{j+1} - p + 1 \dots g_{j+1} - 1]$ is a suffix of G_j and hence it is a prefix-suffix that will be processed by our algorithm; see Figure 4.

We assume that $k = \mathcal{O}(\sqrt{n})$, which is sufficient for our main result. We consider $k_1 \times k_2 = \mathcal{O}(k^2)$ pairs $(F_i, F_{i+1}), (G_j, G_{j+1})$ and, by the analysis in Section 3.3, the time required for processing each of them (i.e. finding the longest prefix-suffix and then considering all its borders in $\mathcal{O}(\log n)$ batches) is bounded by the time required to answer $\mathcal{O}(\log n)$ LCE queries, which can be answered in time $\mathcal{O}(k^2 \log n)$ by Lemma 9. By Lemma 9 we have that $k^2 = \mathcal{O}(n)$ LCE queries over a k -substring of S , can be performed in $\mathcal{O}(k^2 \log n)$ time, using this much extra space, after $\mathcal{O}(n)$ -time preprocessing. Hence the total time required is $\mathcal{O}(k^2 \log^3 n)$ after $\tilde{\mathcal{O}}(n)$ -time preprocessing.

4.4 Main Result

By combining the results of Sections 4.1 to 4.3 we obtain the following.

► **Lemma 15.** (k_1, k_2) -SUBSTRING LCS queries can be answered in $\tilde{\mathcal{O}}(n^{2/3} + k^2)$ time, where $k = k_1 + k_2 = \mathcal{O}(\sqrt{n})$, using a data structure that can be constructed in $\tilde{\mathcal{O}}(n)$ time.

We now formalize the time slicing deamortization technique for our purposes.

► **Lemma 16.** Assume that there is a data structure \mathcal{D} over an input string of length n that occupies $\mathcal{O}(s_n)$ space, answers queries for k -substrings in time $\mathcal{O}(q_n(k))$ and can be constructed in time $\mathcal{O}(t_n)$. Assume that $s_n \geq n$ and $q(k, n) \geq k$ is non-decreasing with respect to k . We can then design an algorithm that preprocesses the input string in time $\mathcal{O}(t_n)$ and answers queries dynamically under edit operations in worst-case time $\mathcal{O}(q_n(\kappa))$, where κ is such that $q_n(\kappa) = (t_n + n)/\kappa$, using $\mathcal{O}(s_n)$ space.

By plugging Lemma 15 into Lemma 16 we arrive at our main result.

► **Theorem 17.** FULLY DYNAMIC LCS on two strings, each of length up to n , can be solved in $\tilde{\mathcal{O}}(n^{2/3})$ time per operation, using $\tilde{\mathcal{O}}(n)$ space, after $\tilde{\mathcal{O}}(n)$ -time preprocessing.

5 Applications

We present three applications of our techniques. The fully dynamic algorithm for computing the longest repeat is very similar to the fully dynamic algorithm for LCS.

Another application is a fully dynamic algorithm for the longest palindrome substring which extends the results of [26, 27]. We consider two cases. In the internal case, in which the longest palindrome occurs between edited positions, we use range queries on the set of maximal palindrome substrings of a string (which is known to have linear size). In the cross-substring case, we use the known fact that the lengths of suffix palindromes of a string can be represented as a logarithmic number of arithmetic progressions which lets us use string periodicity similarly as in Section 3.3. We remark that a more efficient algorithm for computing the longest palindrome in a dynamic string has recently been proposed [6].

The authors of [48] presented algorithms for computing a representation of a Lyndon factorization of a prefix of a string and of a suffix of a string in $\tilde{\mathcal{O}}(1)$ time after $\tilde{\mathcal{O}}(n)$ preprocessing. For the prefixes, their solution is based on the Lyndon representations of prefixes of a Lyndon string, whereas for the suffixes, it is based on the structure of a Lyndon tree (originally due to [13]). In order to devise our fully dynamic algorithm, we carefully combine these two approaches to obtain general internal computation of a representation of a Lyndon factorization in the same time bounds.

6 Final Remarks

We anticipate that the techniques presented in this paper to obtain fully dynamic algorithms for several classical problems on strings are applicable in a wider range of problems on strings.

The significance of our results is additionally highlighted by the following argument. It is known that finding an LCS when the strings have wildcard characters [2] or when $k = \Omega(\log n)$ mismatches are allowed [36] in strongly subquadratic time would refute the Strong Exponential Time Hypothesis (SETH) [33, 32] (on the other hand, pattern matching with wildcard characters can be solved in $\tilde{O}(n)$ time [17] and with k mismatches in $\tilde{O}(n\sqrt{k})$ time [9]). It is therefore unlikely that a fully dynamic algorithm with strongly sublinear-time queries exists for these problems: such an algorithm could be trivially applied as a black box to solve the problems in their static setting in strongly subquadratic time, refuting SETH.

This research could inspire more work on the lower bound side of dynamic problems. The currently known hardness (and conditional hardness) results for dynamic problems on strings have been established for dynamic pattern matching [18, 30]. It would be interesting to investigate (conditional) lower bounds for the dynamic problems considered in this paper.

References

- 1 Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More Applications of the Polynomial Method to Algorithm Design. 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 218–230. SIAM, 2015. doi:10.1137/1.9781611973730.17.
- 2 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of Faster Alignment of Sequences. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2014. doi:10.1007/978-3-662-43948-7_4.
- 3 Paniz Abedin, Sahar Hooshmand, Arnab Ganguly, and Sharma V. Thankachan. The Heaviest Induced Ancestors Problem Revisited. In Gonzalo Navarro, David Sankoff, and Binhai Zhu, editors, *Annual Symposium on Combinatorial Pattern Matching, CPM 2018, July 2-4, 2018 - Qingdao, China*, volume 105 of *LIPICs*, pages 20:1–20:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPICs.CPM.2018.20.
- 4 Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. Pattern Matching in Dynamic Texts. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '00*, pages 819–828, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=338219.338645>.
- 5 Amihod Amir and Itai Boneh. Locally Maximal Common Factors as a Tool for Efficient Dynamic String Algorithms. In Gonzalo Navarro, David Sankoff, and Binhai Zhu, editors, *Annual Symposium on Combinatorial Pattern Matching, CPM 2018, July 2-4, 2018 - Qingdao, China*, volume 105 of *LIPICs*, pages 11:1–11:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPICs.CPM.2018.11.
- 6 Amihod Amir and Itai Boneh. Dynamic Palindrome Detection. *CoRR*, abs/1906.09732, 2019. arXiv:1906.09732.
- 7 Amihod Amir, Itai Boneh, Panagiotis Charalampopoulos, and Eitan Konradovsky. Repetition Detection in a Dynamic String. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-13, 2019, Munich, Germany*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019.

- 8 Amihood Amir, Panagiotis Charalampopoulos, Costas S. Iliopoulos, Solon P. Pissis, and Jakub Radoszewski. Longest Common Factor After One Edit Operation. In Gabriele Fici, Marinella Sciortino, and Rossano Venturini, editors, *String Processing and Information Retrieval: 24th International Symposium, SPIRE 2017, Palermo, Italy, September 26–29, 2017, Proceedings*, volume 10508 of *Lecture Notes in Computer Science*, pages 14–26. Springer International Publishing, 2017. doi:10.1007/978-3-319-67428-5_2.
- 9 Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *Journal of Algorithms*, 50(2):257–275, 2004. doi:10.1016/S0196-6774(03)00097-X.
- 10 Amihood Amir, Moshe Lewenstein, and Sharma V. Thankachan. Range LCP Queries Revisited. In Costas S. Iliopoulos, Simon J. Puglisi, and Emine Yilmaz, editors, *String Processing and Information Retrieval - 22nd International Symposium, SPIRE 2015, London, UK, September 1-4, 2015, Proceedings*, volume 9309 of *Lecture Notes in Computer Science*, pages 350–361. Springer, 2015. doi:10.1007/978-3-319-23826-5_33.
- 11 Alberto Apostolico, Maxime Crochemore, Martin Farach-Colton, Zvi Galil, and S. Muthukrishnan. Forty Years of Text Indexing. In Johannes Fischer and Peter Sanders, editors, *Combinatorial Pattern Matching, 24th Annual Symposium, CPM 2013, Bad Herrenalb, Germany, June 17-19, 2013. Proceedings*, volume 7922 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2013. doi:10.1007/978-3-642-38905-4_1.
- 12 Lorraine A. K. Ayad, Carl Barton, Panagiotis Charalampopoulos, Costas S. Iliopoulos, and Solon P. Pissis. Longest Common Prefixes with k -Errors and Applications. In Travis Gagie, Alistair Moffat, Gonzalo Navarro, and Ernesto Cuadros-Vargas, editors, *String Processing and Information Retrieval - 25th International Symposium, SPIRE 2018, Lima, Peru, October 9-11, 2018, Proceedings*, volume 11147 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2018. doi:10.1007/978-3-030-00479-8_3.
- 13 Hélène Barcelo. On the action of the symmetric group on the Free Lie Algebra and the partition lattice. *Journal of Combinatorial Theory, Series A*, 55(1):93–129, 1990. doi:10.1016/0097-3165(90)90050-7.
- 14 Michael A. Bender and Martin Farach-Colton. The LCA Problem Revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.
- 15 Stefan Burkhardt and Juha Kärkkäinen. Fast Lightweight Suffix Array Construction and Checking. In Ricardo A. Baeza-Yates, Edgar Chávez, and Maxime Crochemore, editors, *Combinatorial Pattern Matching, CPM 2003, Morelia, Michocán, Mexico, June 25–27, 2003*, volume 2676 of *Lecture Notes in Computer Science*, pages 55–69. Springer, 2003. doi:10.1007/3-540-44888-8_5.
- 16 Panagiotis Charalampopoulos, Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Linear-Time Algorithm for Long LCF with k Mismatches. In Gonzalo Navarro, David Sankoff, and Binhai Zhu, editors, *Annual Symposium on Combinatorial Pattern Matching, CPM 2018, July 2-4, 2018 - Qingdao, China*, volume 105 of *LIPICs*, pages 23:1–23:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPICs.CPM.2018.23.
- 17 Peter Clifford and Raphaël Clifford. Simple deterministic wildcard matching. *Information Processing Letters*, 101(2):53–54, 2007. doi:10.1016/j.ipl.2006.08.002.
- 18 Raphaël Clifford, Allan Grönlund, Kasper Green Larsen, and Tatiana A. Starikovskaya. Upper and Lower Bounds for Dynamic Data Structures on Strings. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPICs*, pages 22:1–22:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPICs.STACS.2018.22.

- 19 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007.
- 20 Maxime Crochemore, Costas S. Iliopoulos, Manal Mohamed, and Marie-France Sagot. Longest repeats with a block of k don't cares. *Theoretical Computer Science*, 362(1-3):248–254, 2006. doi:10.1016/j.tcs.2006.06.029.
- 21 Martin Farach. Optimal Suffix Tree Construction with Large Alphabets. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 137–143. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646102.
- 22 Paolo Ferragina. Dynamic Text Indexing under String Updates. *Journal of Algorithms*, 22(2):296–328, 1997. doi:10.1006/jagm.1996.0814.
- 23 Paolo Ferragina and Roberto Grossi. Optimal On-Line Search and Sublinear Time Update in String Matching. *SIAM Journal on Computing*, 27(3):713–736, 1998. doi:10.1137/S0097539795286119.
- 24 Johannes Fischer, Dominik Köppl, and Florian Kurpicz. On the Benefit of Merging Suffix Array Intervals for Parallel Pattern Matching. In Roberto Grossi and Moshe Lewenstein, editors, *27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016, June 27-29, 2016, Tel Aviv, Israel*, volume 54 of *LIPICs*, pages 26:1–26:11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPICs.CPM.2016.26.
- 25 Tomás Flouri, Emanuele Giaquinta, Kassian Kobert, and Esko Ukkonen. Longest common substrings with k mismatches. *Information Processing Letters*, 115(6-8):643–647, 2015.
- 26 Mitsuru Funakoshi, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Longest substring palindrome after edit. In Gonzalo Navarro, David Sankoff, and Binhai Zhu, editors, *Annual Symposium on Combinatorial Pattern Matching, CPM 2018, July 2-4, 2018 - Qingdao, China*, volume 105 of *LIPICs*, pages 12:1–12:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPICs.CPM.2018.12.
- 27 Mitsuru Funakoshi, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Faster Queries for Longest Substring Palindrome After Block Edit. In Nadia Pisanti and Solon P. Pissis, editors, *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy*, volume 128 of *LIPICs*, pages 27:1–27:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPICs.CPM.2019.27.
- 28 Travis Gagie, Paweł Gawrychowski, and Yakov Nekrich. Heaviest Induced Ancestors and Longest Common Substrings. In *Proceedings of the 25th Canadian Conference on Computational Geometry, CCCG 2013, Waterloo, Ontario, Canada, August 8-10, 2013*. Carleton University, Ottawa, Canada, 2013. URL: http://cccg.ca/proceedings/2013/papers/paper_29.pdf.
- 29 Paweł Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Łącki, and Piotr Sankowski. Optimal Dynamic Strings. *CoRR*, abs/1511.02612, 2015. arXiv:1511.02612.
- 30 Paweł Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Łącki, and Piotr Sankowski. Optimal Dynamic Strings. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1509–1528. SIAM, 2018. doi:10.1137/1.9781611975031.99.
- 31 Ming Gu, Martin Farach, and Richard Beigel. An Efficient Algorithm for Dynamic Text Indexing. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '94*, pages 697–704, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=314464.314675>.
- 32 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.
- 33 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Sciences*, 63(4):512–530, December 2001. doi:10.1006/jcss.2001.1774.

- 34 Tomasz Kociumaka. *Efficient Data Structures for Internal Queries in Texts*. PhD thesis, University of Warsaw, October 2018. URL: <https://www.mimuw.edu.pl/~kociumaka/files/phd.pdf>.
- 35 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal Pattern Matching Queries in a Text and Applications. 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 532–551. SIAM, 2015. doi:10.1137/1.9781611973730.36.
- 36 Tomasz Kociumaka, Jakub Radoszewski, and Tatiana A. Starikovskaya. Longest Common Substring with Approximately k Mismatches. *Algorithmica*, 81(6):2633–2652, 2019. doi:10.1007/s00453-019-00548-x.
- 37 Tomasz Kociumaka, Tatiana A. Starikovskaya, and Hjalte Wedel Vildhøj. Sublinear Space Algorithms for the Longest Common Substring Problem. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 605–617. Springer, 2014. doi:10.1007/978-3-662-44777-2_50.
- 38 Roger C. Lyndon. On Burnside’s problem. *Transactions of the American Mathematical Society*, 77:202–215, 1954.
- 39 Mamoru Maekawa. A \sqrt{n} Algorithm for Mutual Exclusion in Decentralized Systems. *ACM Transactions on Computer Systems*, 3(2):145–159, 1985. doi:10.1145/214438.214445.
- 40 Kurt Mehlhorn, R. Sundar, and Christian Urig. Maintaining Dynamic Sequences under Equality Tests in Polylogarithmic Time. *Algorithmica*, 17(2):183–198, 1997. doi:10.1007/BF02522825.
- 41 Süleyman Cenk Sahinalp and Uzi Vishkin. Efficient Approximate and Dynamic Matching of Patterns Using a Labeling Paradigm (extended abstract). In *37th Annual Symposium on Foundations of Computer Science, FOCS ’96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 320–328. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548491.
- 42 Daniel D. Sleator and Robert Endre Tarjan. A Data Structure for Dynamic Trees. *Journal of Computer and System Sciences*, 26(3):362–391, June 1983. doi:10.1016/0022-0000(83)90006-5.
- 43 Tatiana A. Starikovskaya. Longest Common Substring with Approximately k Mismatches. In Roberto Grossi and Moshe Lewenstein, editors, *27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016, June 27-29, 2016, Tel Aviv, Israel*, volume 54 of *LIPICs*, pages 21:1–21:11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPICs.CPM.2016.21.
- 44 Tatiana A. Starikovskaya and Hjalte Wedel Vildhøj. Time-Space Trade-Offs for the Longest Common Substring Problem. In Johannes Fischer and Peter Sanders, editors, *Combinatorial Pattern Matching, 24th Annual Symposium, CPM 2013, Bad Herrenalb, Germany, June 17-19, 2013. Proceedings*, volume 7922 of *Lecture Notes in Computer Science*, pages 223–234. Springer, 2013. doi:10.1007/978-3-642-38905-4_22.
- 45 Rajamani Sundar and Robert Endre Tarjan. Unique Binary-Search-Tree Representations and Equality Testing of Sets and Sequences. *SIAM Journal on Computing*, 23(1):24–44, 1994. doi:10.1137/S0097539790189733.
- 46 Sharma V. Thankachan, Chaitanya Aluru, Sriram P. Chockalingam, and Srinivas Aluru. Algorithmic Framework for Approximate Matching Under Bounded Edits with Applications to Sequence Analysis. In Benjamin J. Raphael, editor, *Research in Computational Molecular Biology - 22nd Annual International Conference, RECOMB 2018, Paris, France, April 21-24, 2018, Proceedings*, volume 10812 of *Lecture Notes in Computer Science*, pages 211–224. Springer, 2018. doi:10.1007/978-3-319-89929-9_14.
- 47 Sharma V. Thankachan, Alberto Apostolico, and Srinivas Aluru. A Provably Efficient Algorithm for the k -Mismatch Average Common Substring Problem. *Journal of Computational Biology*, 23(6):472–482, 2016. doi:10.1089/cmb.2015.0235.

- 48 Yuki Urabe, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Longest Lyndon Substring After Edit. In Gonzalo Navarro, David Sankoff, and Binhai Zhu, editors, *Annual Symposium on Combinatorial Pattern Matching, CPM 2018, July 2-4, 2018 – Qingdao, China*, volume 105 of *LIPICs*, pages 19:1–19:10. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPICs.CPM.2018.19.
- 49 Peter Weiner. Linear Pattern Matching Algorithms. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 1–11. IEEE Computer Society, 1973. doi:10.1109/SWAT.1973.13.

Bilu-Linial Stability, Certified Algorithms and the Independent Set Problem

Haris Angelidakis

ETH Zürich, Switzerland

Pranjal Awasthi

Rutgers University, New Brunswick, NJ, USA

Avrim Blum

Toyota Technological Institute at Chicago, IL, USA

Vaggos Chatziafratis

Stanford University, CA, USA

Chen Dan

Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

We study the classic Maximum Independent Set problem under the notion of *stability* introduced by Bilu and Linial (2010): a weighted instance of Independent Set is γ -stable if it has a unique optimal solution that remains the unique optimal solution under multiplicative perturbations of the weights by a factor of at most $\gamma \geq 1$. The goal then is to efficiently recover this “pronounced” optimal solution exactly. In this work, we solve stable instances of Independent Set on several classes of graphs: we improve upon previous results by solving $\tilde{O}(\Delta/\sqrt{\log \Delta})$ -stable instances on graphs of maximum degree Δ , $(k-1)$ -stable instances on k -colorable graphs and $(1+\varepsilon)$ -stable instances on planar graphs (for any fixed $\varepsilon > 0$), using both combinatorial techniques as well as LPs and the Sherali-Adams hierarchy.

For general graphs, we present a strong lower bound showing that there are no efficient algorithms for $O(n^{\frac{1}{2}-\varepsilon})$ -stable instances of Independent Set, assuming the planted clique conjecture. To complement our negative result, we give an algorithm for (εn) -stable instances, for any fixed $\varepsilon > 0$. As a by-product of our techniques, we give algorithms as well as lower bounds for stable instances of Node Multiway Cut (a generalization of Edge Multiway Cut), by exploiting its connections to Vertex Cover. Furthermore, we prove a general structural result showing that the integrality gap of convex relaxations of several maximization problems reduces dramatically on stable instances.

Moreover, we initiate the study of *certified* algorithms for Independent Set. The notion of a γ -certified algorithm was introduced very recently by Makarychev and Makarychev (2018) and it is a class of γ -approximation algorithms that satisfy one crucial property: the solution returned is optimal for a perturbation of the original instance, where perturbations are again multiplicative up to a factor of $\gamma \geq 1$ (hence, such algorithms not only solve γ -stable instances optimally, but also have guarantees even on unstable instances). Here, we obtain Δ -certified algorithms for Independent Set on graphs of maximum degree Δ , and $(1+\varepsilon)$ -certified algorithms on planar graphs. Finally, we analyze the algorithm of Berman and Fürer (1994) and prove that it is a $(\frac{\Delta+1}{3} + \varepsilon)$ -certified algorithm for Independent Set on graphs of maximum degree Δ where all weights are equal to 1.

2012 ACM Subject Classification Theory of computation \rightarrow Packing and covering problems; Theory of computation \rightarrow Algorithm design techniques

Keywords and phrases Bilu-Linial stability, perturbation resilience, beyond worst-case analysis, Independent Set, Vertex Cover, Multiway Cut

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.7

Related Version A full version of the paper is available at <https://arxiv.org/abs/1810.08414>.

Funding *Haris Angelidakis*: Part of this work was done while the author was a student at TTI-Chicago, and was supported by the National Science Foundation under the grant CCF-1718820.

Avrim Blum: This work was supported in part by the National Science Foundation under grants CCF-1733556, CCF-1800317, and CCF-1815011.



© Haris Angelidakis, Pranjal Awasthi, Avrim Blum, Vaggos Chatziafratis, and Chen Dan; licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 7; pp. 7:1–7:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We would like to thank Konstantin Makarychev and Yury Makarychev for sharing their manuscript [41], and Yury Makarychev and Mrinalkanti Ghosh for useful discussions.

1 Introduction

The Maximum Independent Set problem (simply MIS from now on) is a central problem in theoretical computer science and has been the subject of extensive research over the last few decades. As a result we now have a thorough understanding of the *worst-case* behavior of the problem. In general graphs, the problem is $n^{1-\varepsilon}$ -hard to approximate, assuming that $P \neq NP$ [34, 48], and $n/2^{(\log n)^{3/4+\varepsilon}}$ -hard to approximate, assuming that $NP \not\subseteq \text{BPTIME}(2^{(\log n)^{O(1)}})$ [37]. On the positive side, the current best algorithm is due to Feige [27] achieving a $\tilde{O}(n/\log^3 n)$ -approximation (the notation \tilde{O} hides some $\text{poly}(\log \log n)$ factors). In order to circumvent the strong lower bounds, many works have focused on special classes of graphs, such as bounded-degree graphs (see, e.g., [1, 5, 10, 11, 21, 31, 32, 33]), planar graphs ([7]) etc. In this work, we build upon this long line of research and study MIS under the beyond worst-case framework introduced by Bilu and Linial [17].

In an attempt to capture real-life instances of combinatorial optimization problems, Bilu and Linial proposed a notion of stability, which we now instantiate in the context of MIS (from now on, we will always assume weighted instances of MIS).

► **Definition 1** (γ -perturbation [17]). *Let $G = (V, E, w)$, $w : V \rightarrow \mathbb{R}_{>0}$, be an instance of MIS. An instance $G' = (V, E, w')$ is a γ -perturbation of G , for some parameter $\gamma \geq 1$, if for every $u \in V$ we have $w_u \leq w'_u \leq \gamma \cdot w_u$.*

► **Definition 2** (γ -stability [17]). *Let $G = (V, E, w)$, $w : V \rightarrow \mathbb{R}_{>0}$, be an instance of MIS. The instance G is γ -stable, for some parameter $\gamma \geq 1$, if:*

1. *it has a unique maximum independent set I^* ,*
 2. *every γ -perturbation G' of G has a unique maximum independent set equal to I^* .*
- Equivalently, G is γ -stable if it has an independent set I^* such that $w(I^* \setminus S) > \gamma \cdot w(S \setminus I^*)$ for every feasible independent set $S \neq I^*$ (we use the notation $w(Q) := \sum_{u \in Q} w_u$ for $Q \subseteq V$).*

This definition of stability is motivated by the empirical observation that in many real-life instances, the optimal solution stands out from the rest of the solution space, and thus is not sensitive to small perturbations of the parameters. This suggests that the optimal solution does not change (structurally) if the parameters of the instance are perturbed (even adversarially). Observe that the smaller the so-called *stability threshold* γ is, the less severe the restrictions imposed on the instance are; for example, $\gamma = 1$ is the case where we only require the optimal solution to be unique. Thus, the main goal in this framework is to recover the optimal solution in polynomial time, for as small $\gamma \geq 1$ as possible. An “optimal” result would translate to γ being $1 + \varepsilon$, for small $\varepsilon > 0$, since assuming uniqueness of the optimal solution is not believed to make the problems easier (see, e.g., [47]), and thus ε is unlikely to be zero. We note that perturbations are scale-invariant, and so it suffices to consider perturbations that only scale up. Moreover, we observe that an algorithm for γ -stable instances of MIS solves γ -stable instances of Minimum Vertex Cover, and vice versa.

Stability was first introduced for Max Cut [17], but the authors note that it naturally extends to other problems, such as MIS, and, moreover, they prove that the greedy algorithm for MIS solves Δ -stable instances on graphs of maximum degree Δ . The work of Bilu and Linial has inspired numerous works on stable instances of various optimization problems; we give an overview of the literature in the next page.

Prior works on stability have also studied *robust* algorithms [42, 4]; these are algorithms that either output an optimal solution or provide a polynomial-time verifiable certificate that the instance is not γ -stable (see Section 2 for a definition). Motivated by the notion of stability, Makarychev and Makarychev [41] recently introduced an intriguing class of algorithms, namely γ -certified algorithms.

► **Definition 3** (γ -certified algorithm [41]). *An algorithm for MIS is called γ -certified, for some parameter $\gamma \geq 1$, if for every instance $G = (V, E, w)$, $w : V \rightarrow \mathbb{R}_{>0}$, it computes*

1. *a feasible independent set $S \subseteq V$ of G ,*
 2. *a γ -perturbation $G' = (V, E, w')$ of G such that S is a maximum independent set of G' .*
- Equivalently, Condition (2) can be replaced by the following: $\gamma \cdot w(S \setminus I) \geq w(I \setminus S)$ for every independent set I of G .*

We highlight that a certified algorithm works for *every* instance; if the instance is γ -stable, then the solution returned is the optimal one, while if it is not stable, the solution is within a γ -factor of optimal. Hence a γ -certified algorithm is also a γ -approximation algorithm.

Motivation. Stability is especially natural for problems where the given objective function may be a proxy for a true goal of identifying a hidden correct solution. For MIS, a natural such scenario is applying a machine learning algorithm in the presence of pairwise constraints. Consider, for instance, an algorithm that scans news articles on the web and aims to extract events such as “athlete X won the Olympic gold medal in Y”. For each such statement, the algorithm gives a confidence score (e.g., it might be more confident if it saw this listed in a table rather than inferring it from a free-text sentence that the algorithm might have misunderstood). But in addition, the algorithm might also know logical constraints such as “at most one person can win a gold medal in any given event”. These logical constraints would then become edges in a graph, and the goal of finding the most likely combination of events would become a MIS problem. Stability would be natural to assume in such a setting since the exact confidence weights are somewhat heuristic, and the goal is to recover an underlying ground truth. It is also easy to see the usefulness of a certified algorithm in this setting. Given a certified algorithm that outputs a γ -perturbation, the user of the machine learning algorithm can further test and debug the system by trying to gather evidence for events on which the perturbation puts higher weight.

Related Work. There have been many works on the worst-case complexity of MIS and the current best known algorithms give $\tilde{O}(n/\log^3 n)$ -approximation [27], and $\tilde{O}(\Delta/\log \Delta)$ -approximation [31, 33, 36]), where Δ is the maximum degree. The problem has also been studied from the lens of beyond worst-case analysis. For random graphs with a planted independent set, MIS is equivalent to the classic planted clique problem. Inspired by semi-random models of [18], Feige and Killian [28] designed SDP-based algorithms for computing large independent sets in semi-random graphs. Finally, there has been work on MIS under noise [40, 12].

The notion of Bilu-Linial stability goes beyond random/semi-random models and proposes deterministic conditions that give rise to non worst-case, real-life instances. The study of this notion has led to insights into the complexity of many problems in optimization and machine learning. For MIS, Bilu [15] analyzed the greedy algorithm and showed that it recovers the optimal solution for Δ -stable instances of graphs of maximum degree Δ . The same result is also a corollary of a general theorem about the greedy algorithm and p -extendible independence systems proved by Chatziafratis et al. [22]. On the negative side, Angelidakis et al. [4] showed that there is no robust algorithm for $n^{1-\varepsilon}$ -stable instances of MIS on general graphs (unbounded degree), assuming that $P \neq NP$.

The work of Bilu and Linial has inspired a sequence of works about stable instances of various combinatorial optimization problems. There are now algorithms that solve $O(\sqrt{\log n} \log \log n)$ -stable instances of Max Cut [16, 42], $(2 - 2/k)$ -stable instances of Edge Multiway Cut, where k is the number of terminals [42, 4], and 1.8-stable instances of symmetric TSP [45]. There has also been extensive work on stable instances of clustering problems (usually called perturbation-resilient instances) with many positive results for problems such as k -median, k -means, and k -center [6, 9, 8, 4, 25, 23, 26, 30], and more recently on MAP inference [38, 39].

Our results. We explore the notion of stability in the context of MIS and significantly improve our understanding of its behavior on stable instances; we design algorithms for stable instances on different graph classes, and also initiate the study of certified algorithms for MIS. More specifically, we obtain the following results.

- **Planar graphs:** We show that on planar graphs, any constant stability suffices to solve the problem exactly in polynomial time. More precisely, we provide robust and certified algorithms for $(1 + \varepsilon)$ -stable instances of planar MIS, for any fixed $\varepsilon > 0$. To obtain these results, we utilize the Sherali-Adams hierarchy, demonstrating that hierarchies may be helpful for solving stable instances.
- **Graphs with small chromatic number or bounded degree:** We provide robust algorithms for solving $(k - 1)$ -stable instances of MIS on k -colorable graphs (where the algorithm does not have access to a k -coloring of the graph) and $(\Delta - 1)$ -stable instances of MIS on graphs of maximum degree Δ . Both results are based on LPs. For bounded-degree graphs, we then turn to combinatorial techniques and design a (non-robust) algorithm for $\tilde{O}(\Delta/\sqrt{\log \Delta})$ -stable instances; this is the first algorithm that solves $o(\Delta)$ -stable instances. Moreover, we show that the standard greedy algorithm is a Δ -certified algorithm for MIS, whereas for unweighted instances, the algorithm of Berman and Fürer (1994) is a $(\frac{\Delta+1}{3} + \varepsilon)$ -certified algorithm.
- **General graphs:** For general graphs, we show that solving $o(\sqrt{n})$ -stable instances is hard assuming the hardness of finding maximum cliques in a random graph. To the best of our knowledge, this is only the second case of a lower bound for stable instances of a graph optimization problem that applies to any polynomial-time algorithm and not only to robust algorithms [42, 4] (the first being the lower bound for Max k -Cut [42]). We complement this lower bound by giving an algorithm for (εn) -stable instances of MIS on graphs with n vertices, for any fixed $\varepsilon > 0$.
- **Convex relaxations and stability:** We present a structural result for the integrality gap of convex relaxations of maximization problems on stable instances: if the integrality gap of a relaxation is α , then it is at most $\min\left\{\alpha, 1 + \frac{1}{\beta-1}\right\}$ for $(\alpha\beta)$ -stable instances, for any $\beta > 1$. This result demonstrates a smooth trade-off between stability and the performance of a convex relaxation, and also implies $(1 + \varepsilon)$ -estimation algorithms¹ for $O(\alpha/\varepsilon)$ -stable instances.
- **Node Multiway Cut:** We give the first results on stable instances of Node Multiway Cut, a strict generalization of the well-studied (under stability) Edge Multiway Cut problem [42, 4]. In particular, we give a robust algorithm for $(k - 1)$ -stable instances, where k is the number of terminals, and show that all negative results on stable instances of MIS directly apply to Node Multiway Cut.

¹ An α -estimation algorithm returns a value that is within a factor of α from the optimum, but not necessarily a corresponding solution that realizes this value.

Organization of material. Section 2 provides definitions and related facts. Section 3 contains the algorithms for stable instances of MIS on bounded-degree, small chromatic number and planar graphs. Section 4 contains our results for stable instances on general graphs. Section 5 demonstrates how the performance of convex relaxations improves as stability increases. Section 6 contains various certified algorithms for MIS. Due to space constraints, the results for the Node Multiway Cut problem are omitted and can be found in the full version of the paper [3]. The same applies to all proofs that have been omitted.

2 Preliminaries and definitions

Given a γ -stable instance, our goal is to design polynomial-time algorithms that recover the unique optimal solution, for as small $\gamma \geq 1$ as possible. A special class of such algorithms that is of particular interest is the class of robust algorithms, introduced by Makarychev et al. [42].

► **Definition 4** (robust algorithm [42]). *Let $G = (V, E, w)$, $w : V \rightarrow \mathbb{R}_{>0}$, be an instance of MIS. An algorithm \mathcal{A} is a robust algorithm for γ -stable instances if:*

1. *it always returns the unique optimal solution of G , when G is γ -stable,*
2. *it either returns an optimal solution of G or reports that G is not stable, when G is not γ -stable.*

Note that a robust algorithm is not allowed to err, while a non-robust algorithm is allowed to return a suboptimal solution, if the instance is not γ -stable. We now present a useful lemma about stable instances of MIS that is used in several of our results. From now on, we denote the neighborhood of a vertex u of a graph $G = (V, E)$ as $N(u) = \{v : (u, v) \in E\}$, and the neighborhood of a set $S \subseteq V$ as $N(S) = \{v \in V \setminus S : \exists u \in S \text{ s.t. } (u, v) \in E\}$.

► **Lemma 5.** *Let $G = (V, E, w)$ be a γ -stable instance of MIS whose optimal independent set is I^* . Then, for any $v \in I^*$, the induced instance $\tilde{G} = G[V \setminus (\{v\} \cup N(v))]$ is γ -stable, and its unique maximum independent set is $I^* \setminus \{v\}$.*

Regarding certified algorithms (see Definition 3), it is easy to observe the following.

► **Observation 6** ([41]). *A γ -certified algorithm for MIS satisfies the following:*

1. *returns the unique optimal solution, when run on a γ -stable instance,*
2. *is a γ -certified algorithm for Vertex Cover, and vice versa,*
3. *is a γ -approximation algorithm for MIS (and Vertex Cover).*

We stress that not all algorithms for stable instances are certified, so there is no equivalence between the two notions. Some examples (communicated to us by Yury Makarychev [43]) include the algorithms for stable instances of TSP [45], Max Cut (the GW SDP with triangle inequalities), and clustering. All these algorithms solve stable instances but are not certified. Thus, designing a certified algorithm is, potentially, a harder task than designing an algorithm for stable instances.

From now on, if an algorithm for MIS only returns a feasible solution S , it will be assumed to be “candidate” γ -certified that also returns the perturbed weight function w' with $w'_u = \gamma \cdot w_u$ for $u \in S$ and $w'_u = w_u$, otherwise.

3 Stable instances of MIS on special classes of graphs

In the next few sections, we obtain algorithms for stable instances of MIS on several natural classes of graphs, by using convex relaxations and combinatorial techniques.

3.1 Convex relaxations and robust algorithms

The starting point for the design of robust algorithms via convex relaxations is the structural result of Makarychev et al. [42], that gives sufficient conditions for the integrality of convex relaxations on stable instances. We now introduce a definition and restate their theorem in the setting of MIS.

► **Definition 7** ((α, β) -rounding). *Let $x : V \rightarrow [0, 1]$ be a feasible fractional solution of a convex relaxation of MIS whose objective value for an instance $G = (V, E, w)$ is $\sum_{u \in V} w_u x_u$. A randomized rounding scheme for x is an (α, β) -rounding, for some parameters $\alpha, \beta \geq 1$, if it always returns a feasible independent set S , such that the following two properties hold for every vertex $u \in V$:*

1. $\Pr[u \in S] \geq \frac{1}{\alpha} \cdot x_u$,
2. $\Pr[u \notin S] \leq \beta \cdot (1 - x_u)$.

► **Theorem 8** ([42]). *Let $x : V \rightarrow [0, 1]$ be an optimal fractional solution of a convex relaxation of MIS whose objective value for an instance $G = (V, E, w)$ is $\sum_{u \in V} w_u x_u$. Suppose that there exists an (α, β) -rounding for x , for some $\alpha, \beta \geq 1$. Then, x is integral for (α, β) -stable instances.*

The theorem suggests a simple robust algorithm: solve the relaxation, and if the solution is integral, report it, otherwise report that the instance is not stable (observe that the rounding scheme is used only in the analysis).

3.2 A robust algorithm for $(k - 1)$ -stable instances of MIS on k -colorable graphs

In this section, we give a robust algorithm for $(k - 1)$ -stable instances of MIS on k -colorable graphs by utilizing Theorem 8 and the standard LP for MIS. For a graph $G = (V, E, w)$, the standard LP has an indicator variable x_u for each vertex $u \in V$, and is given below.

$$(\text{LP}) \quad \max : \sum_{u \in V} w_u x_u \quad \text{s.t.} : \quad x_u + x_v \leq 1 \quad \forall (u, v) \in E, \quad \text{and} \quad x_u \in [0, 1] \quad \forall u \in V.$$

The corresponding polytope is half-integral [46], and so we always have an optimal solution x with $x_u \in \{0, \frac{1}{2}, 1\}$ for every $u \in V$. This is useful for designing (α, β) -rounding schemes, as it allows us to consider randomized combinatorial algorithms and easily present them as rounding schemes.

The crucial observation that we make is that the rounding scheme in Theorem 8 is only used in the analysis and is not part of the algorithm, and so it can run in super-polynomial time. We also note that the final (polynomial-time) algorithm does not need to have a k -coloring of the graph. Let $G = (V, E, w)$ be a k -colorable graph, and let x be an optimal half-integral solution. Let $V_i = \{u \in V : x_u = i\}$ for $i \in \{0, 1/2, 1\}$. We consider the rounding scheme of Hochbaum [35] (see Algorithm 1). We use the notation $[k] = \{1, \dots, k\}$.

■ **Algorithm 1** Hochbaum's k -colorable rounding scheme.

-
1. Compute a k -coloring $f : V_{1/2} \rightarrow [k]$ of the induced graph $G[V_{1/2}]$.
 2. Pick j uniformly at random from the set $[k]$, and set $V_{1/2}^{(j)} := \{u \in V_{1/2} : f(u) = j\}$.
 3. Return $S := V_{1/2}^{(j)} \cup V_1$.
-

► **Theorem 9.** *Let $G = (V, E, w)$ be a k -colorable graph. Given an optimal half-integral solution x , the rounding scheme of Algorithm 1 is a $\left(\frac{k}{2}, \frac{2(k-1)}{k}\right)$ -rounding for x .*

Proof. The set S is feasible, as there is no edge between V_1 and $V_{1/2}$ and f is a valid coloring. For $u \in V_0$, we have $\Pr[u \in S] = 0 = x_u$ and $\Pr[u \notin S] = 1 = 1 - x_u$. For $u \in V_1$, we have $\Pr[u \in S] = 1 = x_u$ and $\Pr[u \notin S] = 0 = 1 - x_u$. Let $u \in V_{1/2}$. We have $\Pr[u \in S] \geq \frac{1}{k} = \frac{2}{k} \cdot x_u$ and $\Pr[u \notin S] \leq 1 - \frac{1}{k} = \frac{2(k-1)}{k} \cdot (1 - x_u)$. The result follows. ◀

Theorems 8 and 9 now imply the following theorem, which is tight.

► **Theorem 10.** *The standard LP for MIS is integral for $(k-1)$ -stable instances of k -colorable graphs.*

3.3 Algorithms for stable instances of MIS on bounded-degree graphs

Throughout this section, we assume that all graphs have maximum degree Δ . The only result (prior to our work) for stable instances on such graphs was using the greedy algorithm and was given by Bilu [15].

■ **Algorithm 2** The greedy algorithm for MIS.

-
1. Let $S := \emptyset$ and $X := V$.
 2. while $(X \neq \emptyset)$:
 Set $S := S \cup \{u\}$ and $X := X \setminus (\{u\} \cup N(u))$, where $u := \arg \max_{v \in X} \{w_v\}$.
 3. Return S .
-

► **Theorem 11** ([15]). *The greedy algorithm (see Algorithm 2) solves Δ -stable instances of MIS on graphs of maximum degree Δ .*

We first note that, since the maximum degree is Δ , the chromatic number is at most $\Delta + 1$, and so Theorem 10 implies a robust algorithm for Δ -stable instances, giving a robust analog of Bilu's result. In fact, we can slightly improve upon that by using Brook's Theorem [20], which states that the chromatic number is at most Δ , unless the graph is complete or an odd cycle. We can then prove following theorem.

► **Theorem 12.** *There exists a robust algorithm for $(\Delta - 1)$ -stable instances of MIS, where Δ is the maximum degree.*

We now turn to non-robust algorithms and present an algorithm that solves $o(\Delta)$ -stable instances, as long as the weights are polynomially-bounded integers. The core of the algorithm is a procedure that uses an α -approximation algorithm as a black-box in order to recover the optimal solution, when the instance is stable. Let $G = (V, E, w)$ be a graph with $n = |V|$ and $w : V \rightarrow \{1, \dots, \text{poly}(n)\}$. Let \mathcal{A} denote an α -approximation algorithm for MIS. We will give an algorithm for γ -stable instances with $\gamma = \lceil \sqrt{2\Delta\alpha} \rceil$. Note that we can assume that $\alpha \leq \Delta$ and $\gamma \leq \Delta$. These assumptions hold for the rest of this section. Algorithm 3 is the main algorithm, and it uses Algorithm 4 as a subroutine.

To prove the algorithm's correctness, we need some lemmas (see the full version [3] for their proofs).

► **Lemma 13.** *Let $G = (V, E, w)$ be γ -stable, with $w_u \geq 1$, for every $u \in V$. If $w(V) \leq \gamma$, then $E = \emptyset$.*

■ **Algorithm 3** Algorithm for γ -stable instances, where $\gamma = \lceil \sqrt{2\Delta\alpha} \rceil$.

Bounded-Alg($G(V, E, w)$):

1. If $w(V) \leq \gamma$, then return V .
2. Run α -approximation algorithm \mathcal{A} on G to get an independent set I .
3. Let $S := \text{PURIFY}(G, I, \gamma)$.
4. Let $S' := \text{Bounded-Alg}(G[V \setminus (S \cup N(S))])$.
5. Return $S \cup S'$.

■ **Algorithm 4** The PURIFY procedure.

INPUT: Graph $G = (V, E, w)$, independent set $I \subseteq V$ and factor $\gamma \geq 1$.

1. Create a bipartite unweighted graph $G_0 = (L \cup R, E_0)$, where L contains $\gamma \cdot w(u)$ copies of each $u \in I$ and R contains $w(v)$ copies of each $v \in V \setminus I$. The set E_0 is defined as follows: if (u, v) is an edge in G with $u \in I$ and $v \notin I$, then add edges from each copy of u in L to each copy of v in R .
2. Compute a maximum cardinality matching M of G_0 .
3. Return the set of all vertices $u \in I$ that have at least one unmatched copy in L w.r.t. M .

The above lemma justifies Step 1 of Algorithm 3.

► **Lemma 14.** *Let $G = (V, E, w)$ be γ -stable, and let I^* be its maximum independent set. Then $w(I^*) > \frac{\gamma}{2\Delta} \cdot w(V)$.*

► **Lemma 15.** *Let $G = (V, E, w)$ be a γ -stable instance, let I^* be its maximum independent set and let I' be an α -approximate independent set. Then $I^* \cap I' \neq \emptyset$.*

We now analyze the PURIFY procedure (Algorithm 4).

► **Lemma 16.** *Let G be a γ -stable instance that is given as input to the PURIFY procedure (see Algorithm 4), along with an α -approximate independent set I , and let I^* be its maximum independent set. If $I \neq I^*$, then the set S returned by the procedure always satisfies the following two properties:*

1. $S \neq \emptyset$,
2. $S \subseteq I^*$.

Proof. We first prove Property (1). Let's assume that $S = \emptyset$. This means that all vertices in L are matched. By construction, this implies that $\gamma \cdot w(I) \leq w(V \setminus I)$. Since I is an α -approximation, we have that $\gamma \cdot w(I) \geq \frac{\gamma}{\alpha} \cdot w(I^*) > \frac{\gamma \cdot \gamma}{2\Delta\alpha} w(V) \geq \frac{2\Delta\alpha}{2\Delta\alpha} w(V) = w(V)$, where the second inequality is due to Lemma 14. We conclude that $w(V \setminus I) > w(V)$, which is a contradiction. Thus, $S \neq \emptyset$.

We turn to Property (2). Let $A = I \setminus I^*$ and $B = I^* \setminus I$. Let $A_0 \subseteq L$ be the copies of the vertices of set A in G_0 , and let $B_0 \subseteq R$ be the copies of the vertices of set B in G_0 . We will show that for every $Z \subseteq A_0$, we have $|N(Z) \cap B_0| \geq |Z|$. To see this, let $Z \subseteq A_0$, and let $I(Z) \subseteq A$ be the distinct vertices of A whose copies (not necessarily all of them) are included in Z . Since the instance is γ -stable, this implies that the weight of the neighbors $F \subseteq B$ of $I(Z)$ in I^* is strictly larger than $\gamma \cdot w(I(Z))$. By construction, we have that $|Z| \leq \gamma \cdot w(I(Z))$, and the number of vertices in G_0 corresponding to vertices of F is equal to $w(F)$. Moreover, all of these $w(F)$ vertices are connected with at least one vertex in Z , which means that $w(F) = |N(Z) \cap B_0|$. This implies that $|N(Z) \cap B_0| > |Z|$. Thus, Hall's condition is satisfied, and so there exists a perfect matching between the vertices of A_0 and (a subset of the vertices of) B_0 .

We observe now that the neighbors of all vertices in B_0 are only vertices in A_0 and not in $L \setminus A_0$. This means that any maximum matching matches all vertices of A_0 (otherwise, we could increase the size of the matching by matching all vertices in A_0). Thus, $S \subseteq I \cap I^*$. ◀

Putting everything together, and by using the $\tilde{O}(\Delta/\log \Delta)$ -approximation algorithm of Halldórsson [31] or Halperin [33] as a black-box, it is easy to prove the following theorem.

► **Theorem 17.** *Algorithm 3 correctly solves $\lceil \sqrt{2\Delta\alpha} \rceil$ -stable instances in polynomial time. In particular, there is an algorithm that solves $\tilde{O}(\Delta/\sqrt{\log \Delta})$ -stable instances.*

3.4 Robust algorithms for $(1 + \varepsilon)$ -stable instances of MIS on planar graphs

In this section, we design a robust algorithm for $(1 + \varepsilon)$ -stable instances of MIS on planar graphs. Theorem 10 already implies a robust algorithm for 3-stable instances of planar MIS, but we will use the Sherali-Adams hierarchy (denoted as SA from now on) to reduce this threshold down to $1 + \varepsilon$, for any fixed $\varepsilon > 0$. In particular, we show that $O(1/\varepsilon)$ rounds of SA suffice to optimally solve $(1 + \varepsilon)$ -stable planar instances. We will not introduce the SA hierarchy formally, and we refer the reader to the many available surveys about it (see, e.g., [24]). The t -th level of SA for MIS has a variable Y_S for every subset $S \subseteq V$ of size at most $|S| \leq t + 1$, whose intended value is $Y_S = \prod_{u \in S} x_u$, where x_u is the indicator of whether u belongs to the independent set. The relaxation has size $n^{O(t)}$, and thus can be solved in time $n^{O(t)}$.

Our starting point is the work of Magen and Moharrami [40], which gives a SA-based PTAS for MIS on planar graphs, inspired by Baker's technique [7]. In particular, [40] gives a rounding scheme for the $O(t)$ -th round of SA that returns a $(1 + O(1/t))$ -approximation. In this section, we slightly modify and analyze their rounding scheme, and prove that it satisfies the conditions of Theorem 8. For that, we need a theorem of Bienstock and Ozbay [14]. For any subgraph H of a graph $G = (V, E)$, let $V(H)$ denote the set of vertices contained in H .

► **Theorem 18 ([14]).** *Let $t \geq 1$ and Y be a feasible vector for the t -th level SA relaxation of the standard Independent Set LP for a graph G . Then, for any subgraph H of G of treewidth at most t , the vector $(Y_{\{u\}})_{u \in V(H)}$ is a convex combination of independent sets of H .*

The above theorem implies that the t -th level SA polytope is equal to the convex hull of all independent sets of the graph, when the graph has treewidth at most t .

The rounding scheme of Magen and Moharrami [40]. Let $G = (V, E, w)$ be a planar graph and $\{Y_S\}_{S \subseteq V: |S| \leq t+1}$ be an optimal t -th level solution of SA. We denote $Y_{\{u\}}$ as y_u , for any $u \in V$. We first fix a planar embedding of G . V can then be naturally partitioned into sets V_0, V_1, \dots, V_L , for some $L \in \{0, \dots, n-1\}$, where V_0 is the set of vertices in the boundary of the outerface, V_1 is the set of vertices in the boundary of the outerface after V_0 is removed, and so on. Note that for any edge $(u, v) \in E$, we have $u \in V_i$ and $v \in V_j$ with $|i - j| \leq 1$. We will assume that $L \geq 4$, since, otherwise, the graph is at most 4-outerplanar and the problem can then be solved optimally [7].

Following [7], we fix a parameter $k \in \{1, \dots, L\}$, and for every $i \in \{0, \dots, k-1\}$, we define $B(i) = \bigcup_{j \equiv i \pmod{k}} V_j$. We now pick an index $j \in \{0, \dots, k-1\}$ uniformly at random. Let $G_0 = G[V_0 \cup V_1 \dots \cup V_j]$, and for $i \geq 1$, $G_i = G[\bigcup_{q=(i-1)k+j}^{ik+j} V_q]$, where for a subset $X \subseteq V$, $G[X]$ is the induced subgraph on X . Observe that every edge and vertex of G appears in one or two of the subgraphs $\{G_i\}$, and every vertex $u \in V \setminus B(j)$ appears in exactly one G_i .

Magen and Moharrami observe that for every subgraph $G_i = (V(G_i), E(G_i))$, the set of vectors $\{Y_S\}_{S \subseteq V(G_i): |S| \leq t+1}$ is a feasible solution for the t -th level SA relaxation of the graph G_i . This is easy to see, as the LP associated with G_i is weaker than the LP associated with G (on all common variables), since G_i is a subgraph of G , and this extends to SA as well. We need one more observation: a k -outerplanar graph has treewidth at most $3k - 1$ (see [19]). By construction, each G_i is a $(k+1)$ -outerplanar graph. Thus, by setting $t = 3k + 2$, Theorem 18 implies that the vector $\{y_u\}_{u \in V(G_i)}$ can be written as a convex combination of independent sets of G_i . Let p_i be the corresponding distribution of independent sets of G_i , implied by $\{y_u\}_{u \in V(G_i)}$.

We now consider the following rounding scheme. For each G_i , we (independently) sample an independent set S_i of G_i according to p_i . Each vertex $u \in V \setminus B(j)$ belongs to exactly one G_i and is included in the final independent set S if $u \in S_i$. A vertex $u \in B(j)$ might belong to two different graphs G_i, G_{i+1} , and is included in S only if $u \in S_i \cap S_{i+1}$. The algorithm then returns S .

Before analyzing the algorithm, we note that standard tree-decomposition based arguments show that the rounding is constructive (i.e. polynomial-time; this fact is not needed for the algorithm for stable instances of planar MIS, but will be used when designing certified algorithms).

► **Theorem 19.** *The above randomized rounding scheme always returns a feasible independent set S , such that for every vertex $u \in V$,*

1. $\Pr[u \in S] \geq \frac{k-1}{k} \cdot y_u + \frac{1}{k} \cdot y_u^2$,
2. $\Pr[u \notin S] \leq \left(1 + \frac{1}{k}\right) \cdot (1 - y_u)$.

Proof. It is easy to see that S is always a feasible independent set. We now compute the corresponding probabilities. Since the marginal probability of p_i on a vertex $u \in G_i$ is y_u , for any fixed j , for every vertex $u \in V \setminus B(j)$, we have $\Pr[u \in S] = y_u$, and for every vertex $u \in B(j)$, we have $\Pr[u \in S] \geq y_u^2$. Since j is picked uniformly at random, each vertex $u \in V$ belongs to $B(j)$ with probability exactly equal to $\frac{1}{k}$. Thus, we conclude that for every vertex $u \in V$, we have $\Pr[u \in S] \geq \frac{k-1}{k} \cdot y_u + \frac{1}{k} \cdot y_u^2$, and $\Pr[u \notin S] \leq 1 - \left(\frac{k-1}{k} \cdot y_u + \frac{1}{k} \cdot y_u^2\right) = 1 - y_u + \frac{y_u}{k} \cdot (1 - y_u) \leq \left(1 + \frac{1}{k}\right) \cdot (1 - y_u)$. ◀

The above theorem implies that the rounding scheme is a $\left(\frac{k}{k-1}, \frac{k+1}{k}\right)$ -rounding. The following theorem now is a direct consequence of Theorems 8 and 19.

► **Theorem 20.** *For every $\varepsilon > 0$, the SA relaxation of $(3 \lceil \frac{2}{\varepsilon} \rceil + 5) = O(1/\varepsilon)$ rounds is integral for $(1 + \varepsilon)$ -stable instances of MIS on planar graphs.*

4 Stable instances of MIS on general graphs

In this section, we study stable instances of general graphs. We present a strong lower bound on any algorithm (not necessarily robust) that solves $o(\sqrt{n})$ -stable instances. We complement this lower bound with an algorithm that solves (εn) -stable instances in time $n^{O(1/\varepsilon)}$.

4.1 Computational hardness of stable instances of MIS

We show that for general graphs it is unlikely to obtain efficient algorithms for solving γ -stable instances for small values of γ . Our hardness reduction is based on the *planted clique* conjecture [29, 44], which states that finding $o(\sqrt{n})$ sized planted independent sets/cliques in the Erdős-Rényi graph $G(n, \frac{1}{2})$ is computationally hard. Let $G(n, \frac{1}{2}, k)$ denote the

distribution over graphs obtained by sampling a graph from $G(n, \frac{1}{2})$ and then picking a uniformly random subset of k vertices and deleting all edges among them. The conjecture is formally stated below.

► **Conjecture 21.** *Let $0 < \varepsilon < \frac{1}{2}$ be a constant. Suppose that an algorithm \mathcal{A} receives an input graph G that is either sampled from the ensemble $G(n, \frac{1}{2})$ or $G(n, \frac{1}{2}, n^{\frac{1}{2}-\varepsilon})$. Then, no \mathcal{A} that runs in time polynomial in n can decide, with probability at least $\frac{4}{5}$, which ensemble G was sampled from.*

Our lower bound follows from the observation that planted random instances are stable up to high values of γ , and this suffices to imply our main result.

► **Theorem 22.** *Let $\varepsilon > 0$ be a constant and consider a random graph G on n vertices generated by first picking edges according to the Erdős-Rényi model $G(n, \frac{1}{2})$, followed by choosing a set I of vertices of size $n^{\frac{1}{2}-\varepsilon}$, uniformly at random, and deleting all edges inside I . Then, with probability $1 - o(1)$, the resulting instance is a $\Theta(n^{\frac{1}{2}-\varepsilon}/\log n)$ -stable instance of MIS.*

Proof. Let $G = (V, E)$ be the resulting graph (we assume that all weights are set to 1). We start by stating two well-known properties of the graph G that hold with probability $1 - o(1)$ ([2]).

1. For each vertex $u \in V \setminus I$, we have $|N(u) \cap I| \geq \frac{1}{2} \cdot n^{\frac{1}{2}-\varepsilon} (1 \pm o(1))$.
2. The size of the maximum independent set in the graph $G[V \setminus I]$ is at most $\lceil 2(1 \pm o(1)) \log n \rceil$.

Consider any other independent set $S \neq I$. By Property 1, we have that $|I \setminus S| \geq \frac{1}{2} n^{\frac{1}{2}-\varepsilon} (1 - o(1))$. By Property 2, we must have that $|S \setminus I| \leq 2(1 \pm o(1)) \log n$. Hence, $|S| < |I|$ and furthermore, $|I \setminus S| > \gamma \cdot |S \setminus I|$ for $\gamma = \frac{n^{\frac{1}{2}-\varepsilon}}{4 \log n}$. We conclude that the instance is $\left(\frac{n^{\frac{1}{2}-\varepsilon}}{4 \log n}\right)$ -stable. ◀

4.2 An algorithm for (εn) -stable instances

In this section, we design an algorithm for (εn) -stable instances on graphs of n vertices, that runs in time $n^{O(1/\varepsilon)}$; thus, $\varepsilon > 0$ is assumed to be constant. Due to space constraints, we only give an informal description of the algorithm for the special case of $(n/2)$ -stable instances; the algorithm then naturally generalizes to (n/k) -stable instances, for any integer $k \geq 2$. The base case (i.e., $k = 1$) uses the greedy algorithm.

We start by observing that either the chromatic number is at most $n/2$, in which case the LP is integral, or there are more than $n/2$ vertices of degree at least $n/2$. In the latter case, we check whether each of these high-degree vertices belongs to the optimal solution, by removing each such vertex and its neighborhood, one at a time, and using the algorithm recursively. Since their neighborhoods are large, we end up with graphs with at most $n/2$ vertices, which are still $(n/2)$ -stable if the removed vertex belongs to the optimal solution; thus, the recursion succeeds on them (in particular, the greedy algorithm solves such instances). Finally, if none of these high-degree vertices belongs to the optimal solution, then we remove all of them, and end up again with a graph with at most $n/2$ vertices that is $(n/2)$ -stable; the recursive call again solves that instance. By returning the best of all computed solutions, we are guaranteed to recover the optimal solution.

► **Theorem 23.** *There exists an algorithm that solves $(\frac{n}{k})$ -stable instances of MIS on graphs of n vertices in time $n^{O(k)}$.*

5 Stability and integrality gaps of convex relaxations

In this section, we state a general theorem about the integrality gap of convex relaxations of maximization problems on stable instances. In particular, we show that, even if the conditions of Theorem 8 are not satisfied, the integrality gap still significantly decreases as stability increases.

► **Theorem 24.** *Consider a relaxation for MIS that assigns a value $x_u \in [0, 1]$ to each vertex u of a graph $G = (V, E, w)$, and its objective function is $\sum_{u \in V} w_u x_u$. Let α be its integrality gap, for some $\alpha > 1$. Then, its integrality gap is at most $\min\{\alpha, 1 + \frac{1}{\beta-1}\}$ on $(\alpha\beta)$ -stable instances, for any $\beta > 1$.*

The proof is somewhat similar in spirit to the lemmas and analysis used for Algorithm 3. We stress that the above result is inherently non-constructive. Nevertheless, it suggests estimation algorithms for stable instances of MIS, such as the following, which is a direct consequence of Theorem 24 and the results of [31, 33].

► **Corollary 25.** *For any fixed $\varepsilon > 0$, the Lovasz θ -function SDP has integrality gap at most $1 + \varepsilon$ on $\tilde{O}\left(\frac{1}{\varepsilon} \cdot \frac{\Delta}{\log \Delta}\right)$ -stable instances of MIS of maximum degree Δ .*

We note that the theorem naturally extends to many other maximization graph problems, and is particularly interesting for relaxations that require super-constant stability for the recovery of the optimal solution (e.g., the Max Cut SDP has integrality gap $1 + \varepsilon$ for $(2/\varepsilon)$ -stable instances although the integrality gap drops to exactly 1 for $\Omega(\sqrt{\log n} \cdot \log \log n)$ -stable instances).

In general, such a theorem is not expected to hold for minimization problems, but, in our case, MIS gives rise to its complementary Minimum Vertex Cover problem, and it turns out that we can prove a very similar result for Minimum Vertex Cover as well.

6 Certified algorithms for MIS

In this section, we initiate the systematic study of certified algorithms for MIS, introduced by Makarychev and Makarychev [41].

6.1 Certified algorithms using convex relaxations

An important observation that [41] makes is that an approach very similar to the one used for the design of algorithms for weakly-stable instances [42] can be used to obtain certified algorithms. More formally, they prove the following theorem.

► **Theorem 26** ([41]). *Let $x : V \rightarrow [0, 1]$ be an optimal fractional solution of a convex relaxation of MIS whose objective value for an instance $G = (V, E, w)$ is $\sum_{u \in V} w_u x_u$. Suppose that there exists a polynomial-time (α, β) -rounding for x . Then, there exists a polynomial-time $(\alpha\beta + \varepsilon)$ -certified algorithm for MIS on instances with integer polynomially-bounded weights (for $\varepsilon \geq 1/\text{poly}(n) > 0$).*

We now combine Theorem 19 with Theorem 26 and obtain the following theorem.

► **Theorem 27.** *There exists a polynomial-time $(1 + \varepsilon)$ -certified algorithm for MIS on planar graphs with integer polynomially-bounded weights (for $\varepsilon \geq 1/\text{poly}(n) > 0$).*

6.2 Combinatorial certified algorithms

In this section, we study several combinatorial algorithms for MIS and prove that they are certified. The first result is about the greedy algorithm.

► **Theorem 28.** *The greedy algorithm (see Algorithm 2) is a Δ -certified algorithm for MIS on graphs of maximum degree Δ . More generally, the greedy algorithm is a Δ -certified algorithm for any instance of a Δ -extendible system.*

Moreover, we introduce a variant of the greedy algorithm for MIS that is a $\sqrt{\Delta^2 - \Delta + 1}$ -certified algorithm; the improvement over the greedy is moderate for small values of Δ . Finally, we show that the algorithm of Berman and Fürer [13] is $(\frac{\Delta+1}{3} + \varepsilon)$ -certified, when all weights are 1. We acknowledge that the restriction to unweighted graphs limits the scope of the algorithm, but we consider this as a first step towards obtaining $(c\Delta)$ -certified algorithms, for $c < 1$.

► **Theorem 29.** *The Berman-Fürer algorithm ([13]) is a $(\frac{\Delta+1}{3} + \varepsilon)$ -certified algorithm for MIS on graphs of maximum degree Δ , when all weights are equal to 1.*

Let $G = (V, E, w)$ be a graph of maximum degree Δ , $n = |V|$, where $w_u = 1$ for every $u \in V$. We say X is an improvement of I , if both I and $I \oplus X$ are independent sets, the subgraph induced by X is connected and $I \oplus X$ is larger than I . (The operator \oplus denotes the symmetric difference.)

The algorithm starts with a feasible independent set I' and iteratively improves the solution by checking whether there exists an improvement X with size $|X| \leq \sigma$. If so, it replaces I by $I \oplus X$ and repeats. Otherwise, if no such improvement exists, it outputs the current independent set I . Assuming that Δ is a constant, the algorithm runs in polynomial time as long as $\sigma = O(\log n)$.

► **Lemma 30** ([13]). *If Δ is a constant and $\sigma = O(\log n)$, the algorithm runs in polynomial time.*

The main result can be presented as follows. Along with Definition 3, it implies Theorem 29.

► **Lemma 31.** *Let I be the independent set returned by the algorithm with $\sigma = 32k\Delta^{4k} \log n$ and let $S \neq I$ be any feasible independent set. Then, we have $|S \setminus I| \leq (\frac{\Delta+1}{3} + \varepsilon) \cdot |I \setminus S|$, where $\varepsilon = \frac{1}{3k}$.*

Proof. Let $\bar{S} = S \setminus I$ and $\bar{I} = I \setminus S$. First, we observe that every $u \in \bar{S}$ has at least one neighbor in \bar{I} , otherwise, we could improve I by adding a new vertex from \bar{S} . We now consider the set $T = \{u \in \bar{S} : |N(u) \cap I| = 1\} \subseteq \bar{S}$. In words, T is the set of elements in \bar{S} that have exactly one neighbor in I . We also define $J = \{v \in \bar{I} : N(v) \cap T \neq \emptyset\}$ to be the set of elements of \bar{I} that have at least one neighbor in T . We will show that $|T| \leq |J|$.

To prove this, let's assume that $|T| > |J|$. Then, by the pigeonhole principle, we must have at least one vertex $v \in J$ that is connected to at least two vertices $u_1, u_2 \in T$. This implies that replacing v with u_1 and u_2 would be an improvement. Thus, we get a contradiction. Now let $I_0 = \bar{I} \setminus J$ and $S_0 = \bar{S} \setminus T$. The final step of the proof is a direct consequence of Lemma 3.5 of [13], that states that if there is no improvement over I of size at most $\sigma = 32k\Delta^{4k} \log n$, then for $\varepsilon = 1/(3k)$, $|S_0| \leq (\frac{\Delta+1}{3} + \varepsilon) |I_0|$. Recall that we have already proved $|T| \leq |J|$. Therefore,

$$\begin{aligned} |S \setminus I| &= |S_0| + |T| \leq \left(\frac{\Delta+1}{3} + \varepsilon\right) |I_0| + |J| \\ &\leq \left(\frac{\Delta+1}{3} + \varepsilon\right) (|I_0| + |J|) = \left(\frac{\Delta+1}{3} + \varepsilon\right) |I \setminus S|. \end{aligned} \quad \blacktriangleleft$$

References

- 1 Noga Alon and Nabil Kahale. Approximating the independence number via the θ -function. *Math. Program.*, 80:253–264, 1998.
- 2 Noga Alon, Michael Krivelevich, and Benny Sudakov. Finding a large hidden clique in a random graph. *Random Structures and Algorithms*, 13(3-4):457–466, 1998.
- 3 Haris Angelidakis, Pranjali Awasthi, Avrim Blum, Vaggos Chatziafratis, and Chen Dan. Bilu-Linial stability, certified algorithms and the Independent Set problem. *CoRR*, abs/1810.08414, 2018. [arXiv:1810.08414](https://arxiv.org/abs/1810.08414).
- 4 Haris Angelidakis, Konstantin Makarychev, and Yury Makarychev. Algorithms for stable and perturbation-resilient problems. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 438–451, 2017.
- 5 Per Austrin, Subhash Khot, and Muli Safra. Inapproximability of Vertex Cover and Independent Set in Bounded Degree Graphs. *Theory of Computing*, 7(1):27–43, 2011.
- 6 Pranjali Awasthi, Avrim Blum, and Or Sheffet. Center-based clustering under perturbation stability. *Inf. Process. Lett.*, 112(1-2):49–54, 2012.
- 7 Brenda S. Baker. Approximation Algorithms for NP-Complete Problems on Planar Graphs. *J. ACM*, 41(1):153–180, 1994.
- 8 Maria-Florina Balcan, Nika Haghtalab, and Colin White. k -Center Clustering Under Perturbation Resilience. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 68:1–68:14, 2016.
- 9 Maria-Florina Balcan and Yingyu Liang. Clustering under Perturbation Resilience. *SIAM J. Comput.*, 45(1):102–155, 2016.
- 10 Nikhil Bansal. Approximating independent sets in sparse graphs. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–8, 2015.
- 11 Nikhil Bansal, Anupam Gupta, and Guru Guruganesh. On the Lovász Theta function for Independent Sets in Sparse Graphs. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 193–200, 2015.
- 12 Nikhil Bansal, Daniel Reichman, and Seeun William Umboh. LP-based robust algorithms for noisy minor-free and bounded treewidth graphs. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1964–1979, 2017.
- 13 Piotr Berman and Martin Fürer. Approximating Maximum Independent Set in Bounded Degree Graphs. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 365–371, 1994.
- 14 Daniel Bienstock and Nuri Özbay. Tree-width and the Sherali-Adams operator. *Discrete Optimization*, 1(1):13–21, 2004.
- 15 Yonatan Bilu. On spectral properties of graphs and their application to clustering. *PhD Thesis*, pages 77–78, 2004.
- 16 Yonatan Bilu, Amit Daniely, Nati Linial, and Michael E. Saks. On the practically interesting instances of MAXCUT. In *Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 526–537, 2013.
- 17 Yonatan Bilu and Nathan Linial. Are Stable Instances Easy? *Combinatorics, Probability & Computing*, 21(5):643–660, 2012.
- 18 Avrim Blum and Joel Spencer. Coloring random and semi-random k -colorable graphs. *Journal of Algorithms*, 19(2):204–234, 1995.
- 19 Hans L. Bodlaender. A partial k -arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1):1–45, 1998.
- 20 R. L. Brooks. On colouring the nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, 37(2):194–197, 1941.
- 21 Siu On Chan. Approximation Resistance from Pairwise-Independent Subgroups. *J. ACM*, 63(3):27:1–27:32, 2016.

- 22 Vaggos Chatziafratis, Tim Roughgarden, and Jan Vondrák. Stability and Recovery for Independence Systems. In *Proceedings of the 25th Annual European Symposium on Algorithms (ESA)*, pages 26:1–26:15, 2017.
- 23 Chandra Chekuri and Shalmoli Gupta. Perturbation Resilient Clustering for k -Center and Related Problems via LP Relaxations. In *Proceedings of the 21st International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, 2018.
- 24 Eden Chlamtac and Madhur Tulsiani. *Convex Relaxations and Integrality Gaps*, pages 139–169. Springer US, Boston, MA, 2012.
- 25 Vincent Cohen-Addad and Chris Schwiegelshohn. On the Local Structure of Stable Clustering Instances. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 49–60, 2017.
- 26 Amit Deshpande, Anand Louis, and Apoorv Vikram Singh. On Euclidean k -Means Clustering with α -Center Proximity. In *the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 2087–2095, 2019.
- 27 Uriel Feige. Approximating Maximum Clique by Removing Subgraphs. *SIAM J. Discrete Math.*, 18(2):219–225, 2004.
- 28 Uriel Feige and Joe Kilian. Heuristics for semirandom graph problems. *Journal of Computer and System Sciences*, 63(4):639–671, 2001.
- 29 Vitaly Feldman, Elena Grigorescu, Lev Reyzin, Santosh Vempala, and Ying Xiao. Statistical Algorithms and a Lower Bound for Detecting Planted Cliques. *J. ACM*, 64(2):8:1–8:37, 2017.
- 30 Zachary Friggstad, Kamyar Khodamoradi, and Mohammad R. Salavatipour. Exact Algorithms and Lower Bounds for Stable Instances of Euclidean k -MEANS. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2958–2972, 2019.
- 31 Magnús M. Halldórsson. Approximations of Weighted Independent Set and Hereditary Subset Problems. *J. Graph Algorithms Appl.*, 4(1), 2000.
- 32 Magnús M. Halldórsson and Jaikumar Radhakrishnan. Improved Approximations of Independent Sets in Bounded-Degree Graphs via Subgraph Removal. *Nord. J. Comput.*, 1(4):475–492, 1994.
- 33 Eran Halperin. Improved Approximation Algorithms for the Vertex Cover Problem in Graphs and Hypergraphs. *SIAM J. Comput.*, 31(5):1608–1623, 2002.
- 34 Johan Håstad. Clique is Hard to Approximate Within $n^{1-\epsilon}$. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 627–636, 1996.
- 35 Dorit S. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6(3):243–254, 1983.
- 36 Akihisa Kako, Takao Ono, Tomio Hirata, and Magnús M. Halldórsson. Approximation algorithms for the weighted independent set problem in sparse graphs. *Discrete Applied Mathematics*, 157(4):617–626, 2009.
- 37 Subhash Khot and Ashok Kumar Ponnuswami. Better Inapproximability Results for MaxClique, Chromatic Number and Min-3Lin-Deletion. In *Proceedings of the 33rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 226–237, 2006.
- 38 Hunter Lang, David Sontag, and Aravindan Vijayaraghavan. Optimality of Approximate Inference Algorithms on Stable Instances. In *the 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1157–1166, 2018.
- 39 Hunter Lang, David Sontag, and Aravindan Vijayaraghavan. Block Stability for MAP Inference. In *the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 216–225, 2019.
- 40 Avner Magen and Mohammad Moharrami. Robust Algorithms for Maximum Independent Set on Minor-Free Graphs Based on the Sherali-Adams Hierarchy. In *Proceedings of the 12th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 258–271, 2009.
- 41 Konstantin Makarychev and Yury Makarychev. Certified approximation algorithms: Bridging worst-case and beyond-the-worst-case analysis. *Manuscript*, 2018.

- 42 Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Bilu-Linial Stable Instances of Max Cut and Minimum Multiway Cut. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 890–906, 2014.
- 43 Yury Makarychev. Private communication, 2018.
- 44 Raghu Meka, Aaron Potechin, and Avi Wigderson. Sum-of-squares Lower Bounds for Planted Clique. In *Proceedings of the 47th Annual ACM on Symposium on Theory of Computing (STOC)*, pages 87–96, 2015.
- 45 Matús Mihalák, Marcel Schöngens, Rastislav Srámek, and Peter Widmayer. On the Complexity of the Metric TSP under Stability Considerations. In *SOFSEM 2011: Theory and Practice of Computer Science - 37th Conference on Current Trends in Theory and Practice of Computer Science*, pages 382–393, 2011.
- 46 G. L. Nemhauser and L. E. Trotter. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975.
- 47 Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986.
- 48 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 681–690, 2006.

On the Complexity of Anchored Rectangle Packing

Antonios Antoniadis 

Universität des Saarlandes,
Saarbrücken, Germany
Max-Planck-Institut für Informatik,
Saarbrücken, Germany
aantonia@mpi-inf.mpg.de

Andrés Cristi

Universidad de Chile, Santiago, Chile
andres.cristi@ing.uchile.cl

Ruben Hoeksma 

Universität Bremen, Germany
University of Twente, Enschede, The Netherlands
hoeksma@uni-bremen.de

Peter Kling 

Universität Hamburg, Germany
peter.kling@uni-hamburg.de

Felix Biermeier

Universität Hamburg, Germany
felix.biermeier@uni-hamburg.de

Christoph Damerius

Universität Hamburg, Germany
christoph.damerius@uni-hamburg.de

Dominik Kaaser 

Universität Hamburg, Germany
dominik.kaaser@uni-hamburg.de

Lukas Nölke 

Universität Bremen, Germany
noelke@uni-bremen.de

Abstract

In the *Anchored Rectangle Packing (ARP)* problem, we are given a set of points P in the unit square $[0, 1]^2$ and seek a maximum-area set of axis-aligned interior-disjoint rectangles S , each of which is anchored at a point $p \in P$. In the most prominent variant – *Lower-Left-Anchored Rectangle Packing (LLARP)* – rectangles are anchored in their lower-left corner. Freedman [19, Unsolved Problem 11, page 345] conjectured in 1969 that, if $(0, 0) \in P$, then there is a LLARP that covers an area of at least 0.5. Somewhat surprisingly, this conjecture remains open to this day, with the best known result covering an area of 0.091 [11]. Maybe even more surprisingly, it is not known whether LLARP – or any ARP-problem with only one anchor – is NP-hard.

In this work, we first study the *Center-Anchored Rectangle Packing (CARP)* problem, where rectangles are anchored in their center. We prove NP-hardness and provide a PTAS. In fact, our PTAS applies to any ARP problem where the anchor lies in the interior of the rectangles. Afterwards, we turn to the LLARP problem and investigate two different resource-augmentation settings: In the first we allow an ε -perturbation of the input P , whereas in the second we permit an ε -overlap between rectangles. For the former setting, we give an algorithm that covers at least as much area as an optimal solution of the original problem. For the latter, we give an $(1 - \varepsilon)$ -approximation.

2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases anchored rectangle, rectangle packing, resource augmentation, PTAS, NP, hardness

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.8

Acknowledgements We thank the organizers of the Bremen-Hamburg workshop on algorithms, combinatorics, and optimization for organizing the open problem session where we first learned of the Freedman conjecture. In particular, we thank Nicole Megow from University of Bremen for posing the conjecture during that session and for valuable suggestions. Parts of this work were a result of the UHH & UChile TCS Workshop 2018.



© Antonios Antoniadis, Felix Biermeier, Andrés Cristi, Christoph Damerius, Ruben Hoeksma, Dominik Kaaser, Peter Kling, and Lukas Nölke;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 8; pp. 8:1–8:14



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The *lower-left-anchored rectangle packing (LLARP)* problem was first posed in 1969 by Freedman [19]: Given the unit square $U := [0, 1]^2$ in the plane and a finite set of points $P \subseteq U$, find a maximum-area set of axis-aligned and interior-disjoint rectangles \mathcal{S} , such that for each $p \in P$, there is exactly one (possibly trivial) rectangle $R \in \mathcal{S}$ that has p in its lower-left corner. It was conjectured [19, 20] that the area covered is at least 0.5 as long as $(0, 0) \in P$. The first and so far only result that achieves a constant lower bound covers an area of at least 0.091 [11].

A natural generalization of the LLARP problem looks at anchors different from the lower-left corner. Such an *anchoring* can be defined by a pair $(\alpha, \beta) \in [0, 1]^2$. The (α, β) -ARP problem then looks for a maximum-area rectangle set as above but with rectangles anchored in the relative position (α, β) (see Section 2). For example, $(0, 0)$ -ARP is LLARP, while $(1/2, 1/2)$ -ARP requires all rectangles to have a $p \in P$ at their center. We refer to the latter as *center-anchored rectangle packing (CARP)* problem.

Our Contribution. Even though the formulation of LLARP and Freedman’s conjecture [19], date back to 1969 it is still not even known whether its decision variant is NP-hard. This is why, in this work, we focus on the complexity of related ARP problems and introduce a resource augmentation setting for LLARP.

When looking at the complexity of CARP, a difference that stands out compared to LLARP is that CARP allows one to simulate “non-expandable” points: By putting four points at the corners of a tiny ε -sized square, none of their rectangles’ side lengths can exceed ε . One can think of these four points as one input point that cannot be used as an anchor but still restricts the expansion of other rectangles. These non-expandable points turn out to be a valuable asset, as they can be used to build walls and inject additional geometry into the problem. We exploit this in an elaborate construction that encodes maximum independent set into a CARP instance, proving NP-hardness of CARP (cf. Section 4). The construction of non-expandable points seems difficult or even impossible for LLARP and is the main obstacle in transferring our NP-hardness proof to that setting.

The NP-hardness is complemented by a polynomial-time approximation scheme (PTAS), which extends also to any anchoring $(\alpha, \beta) \in (0, 1)^2$. The PTAS is based on a carefully constructed input instance for a related problem called *maximum weight independent set of rectangles (MWISR)* and the usage of a known PTAS in a resource augmentation setting of MWISR (cf. Section 3).

With respect to the classical LLARP problem, we initiate the investigation of the problem with resource augmentation. We study two such settings. In the first, the algorithm is allowed to slightly perturb the input points. In the second, the rectangle set produced by the algorithm may have some (bounded) overlap. In both cases, the resulting solution is compared to an optimal solution without these augmentations. For the first setting, we develop an algorithm that produces an anchored rectangle set of total area no less than that of an optimal solution of LLARP without resource augmentation (cf. Section 5). Our analysis is combinatorial in nature and consists of transforming the optimal solution to a feasible solution for a perturbed instance by using a specific linear program with totally unimodular incidence matrix. For the second setting, we provide an algorithm that covers at least $(1 - \varepsilon)$ times the area of an optimal solution (cf. Section 5). Our algorithm is based on the one for the first setting.

Further Related Work. The best known polynomial time algorithm for the LLARP problem is due to Dumitrescu and Tóth [11] and covers an area of at least 0.091. Since the optimal solution cannot cover more than an area of 1 (the whole unit square), their analysis also implies a 0.091-approximation. In the setting where, instead of arbitrary rectangles, squares must be used, Balas et al. [5] achieve an approximation ratio of $1/3$. They also consider a setting where the algorithm can choose for each rectangle from multiple anchors (the four corners), giving a $(7/12 - \varepsilon)$ -approximation algorithm, where $\varepsilon = |P|^{-1}$, and a QPTAS for rectangles, as well as a $9/47$ -approximation algorithm and a PTAS for squares. The QPTAS and PTAS extend to the lower-left anchored variants. Recently, Akitaya et al. [4] gave the first NP-hardness result for an ARP variant where only squares may be packed and each square can be anchored at any of its four corners. They also show that, for any instance consisting of finitely many input points inside U , the union of all feasible anchored square packings covers an area of at least $1/2$. Finally, if rectangles can be anchored at any of their four corners but input points are restricted to the boundary of U , Biedl et al. [9] give a polynomial-time algorithm (based on maximum independent set for a specific class of graphs).

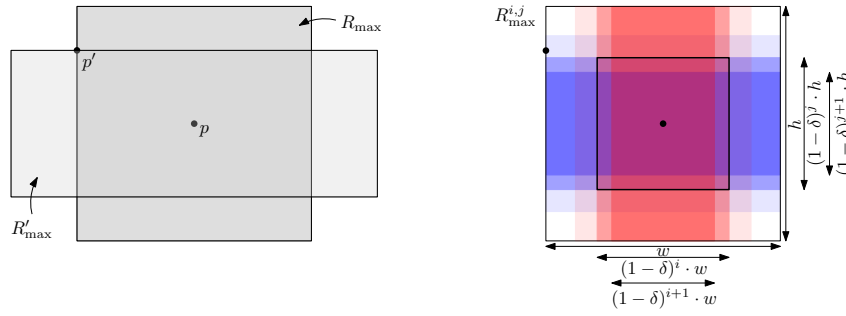
ARP problems fall within the more general setting of packing axis-aligned and interior-disjoint rectangles in a rectangular container. This setting captures several important and well studied optimization problems. See, for example, the (NP-hard) problems 2D-knapsack and strip packing [2, 6], as well as maximum area independent set of rectangles [3, 7, 8]. Similar problems have also been formulated by Radó and Rado [16, 13, 14, 15]. These problems differ from ARP in that the size of the packed objects is part of the input and not controllable and, in some cases, there is no anchoring of the rectangles.

2 Preliminaries

We start with some general notation. Consider a point $p \in \mathbb{R}^2$ and an axis-aligned rectangle $R \subseteq \mathbb{R}^2$ with its lower-left corner in $(x, y) \in \mathbb{R}^2$. Let w and h denote the width and height of R , respectively. For a pair $(\alpha, \beta) \in [0, 1]^2$ (called *anchoring*), we say R is (α, β) -anchored in p if $p = (x + \alpha \cdot w, y + \beta \cdot h)$. We also say that p (α, β) -spans R . If the anchoring (α, β) is clear from the context we omit it. We use $\mathcal{A}(R) = w \cdot h$ to denote the area of rectangle R . Similarly, given a set \mathcal{S} of rectangles we define $\mathcal{A}(\mathcal{S})$ as the area of $\bigcup_{R \in \mathcal{S}} R$. We use $U = [0, 1]^2$ to denote the unit square. If not stated otherwise, any rectangle is assumed to be axis-aligned.

The Anchored Rectangle Packing Problem. We now define the *anchored rectangle packing (ARP)* problem with respect to an anchoring $(\alpha, \beta) \in [0, 1]^2$. An instance consists of a finite set $P \subseteq U$ of n points. A *valid rectangle set* (also *solution*) \mathcal{S} for the ARP instance P is a set of interior-disjoint rectangles that contains one (possibly zero-sized) rectangle R_p for each $p \in P$ such that R_p is (α, β) -anchored in p and does not contain any point from $P \setminus \{p\}$ in its interior. The goal is to find a solution \mathcal{S} for P that covers as much area as possible. We use $\mathcal{S}_{\text{ARP}(P)}^*$ to denote a valid rectangle set of maximum area $\text{OPT}_{\text{ARP}(P)} := \mathcal{A}(\mathcal{S}_{\text{ARP}(P)}^*)$ and omit ARP and/or P if it is clear from context.

We will mostly consider two specific anchorings. When using the anchoring $(0, 0)$, we refer to the problem by the name *lower-left-anchored rectangle packing (LLARP)*. When the anchoring is $(1/2, 1/2)$, we refer to the problem by the name *center-anchored rectangle packing (CARP)*.



■ **Figure 1** Maximal rectangles R_{\max} and R'_{\max} and candidate rectangles $R_{\max}^{i,j}$ derived from R_{\max} for the anchoring $(1/2, 1/2)$.

3 A PTAS for ARP with Fractional Anchorings

In the following, we give a PTAS for the anchored rectangle packing problem for any anchoring $(\alpha, \beta) \in (0, 1)^2$. It is based on a result for *maximum weight independent set of rectangles (MWISR)*¹ that uses resource augmentation similar to that in Theorem 8. An instance of MWISR consists of a set of n axis-aligned, weighted rectangles. The goal is to compute a maximum-weight subset of pairwise interior-disjoint rectangles. In the relaxed variant δ -MWISR (for fixed $\delta > 0$), rectangles must be non-overlapping only *after* shrinking them by a factor of $1 - \delta$. Here, Adamaszek et al. [1] give an algorithm that, in time $n^{(\delta\varepsilon)^{-\Omega(1/\varepsilon)}}$, computes a solution which is within a $(1 + \varepsilon)$ -factor of the optimum MWISR. While the original algorithm considers shrinking only around the rectangles' centers, this can be generalized such that it also works when shrinking around an arbitrary anchoring $(\alpha, \beta) \in (0, 1)^2$. However, this does not generalize to anchors on the boundaries (in particular, not to LLARP), since [1] requires shrinking around a rectangle's center, this way the shrunken rectangle would no longer contain its anchor.

► **Theorem 1.** *For any fixed anchoring $(\alpha, \beta) \in (0, 1)^2$, the anchored rectangle packing problem admits a polynomial-time approximation scheme.*

Proof. Consider an instance $P \subseteq U$ of (α, β) -ARP with n input-points and fix $\varepsilon > 0$. Denote by ε', δ positive constants which we fix later and define $N := \lceil \log_{1-\delta}(\delta/n) \rceil + 1$.

For each point pair $p, p' \in P$, there are at most two inclusion-wise maximal rectangles that are anchored in p and have p' on their boundary, one where p' is on the left (right) side of the rectangle and one where p' is on its bottom (top) side. Thus, there are at most $2n$ inclusion-wise maximal rectangles that are anchored in a point $p \in P$ and at most $2n^2$ overall. For any such *maximal rectangle* R_{\max} , construct N^2 *candidate rectangles* $(R_{\max}^{i,j})_{i,j=0}^{N-1}$ by scaling (around the anchor) by a factor of $(1 - \delta)^i$ along the horizontal and $(1 - \delta)^j$ along the vertical axis. Denote the set of all such candidate rectangles by \mathcal{C} . See Figure 1 for an illustration.

In order to obtain a $(1 + \varepsilon)$ -approximation for the ARP instance P , we first apply the algorithm for δ -MWISR [1] to the MWISR instance $(\mathcal{C}, \mathcal{A})$ (i.e., all candidate rectangles weighted by their area). This yields a rectangle set \mathcal{S}' with $\mathcal{A}(\mathcal{S}') \geq (1 + \varepsilon')^{-1} \cdot \text{OPT}_{\text{MWISR}}$.

¹ A similar connection to MWISR has recently been used [5] to obtain a PTAS for packing squares and a QPTAS for packing rectangles when the anchors are allowed to lie in any of the four corners (and may be different for each rectangle).

where $\text{OPT}_{\text{MWISR}}$ denotes the area covered by an optimal solution for the instance $(\mathcal{C}, \mathcal{A})$ of MWISR. Note that each rectangle $R \in \mathcal{S}'$ is anchored in some point $p \in P$. By definition of δ -MWISR, scaling each rectangle in \mathcal{S}' by a factor of $1 - \delta$ in each dimension around its anchor produces a set \mathcal{S} of pairwise interior-disjoint rectangles with

$$\mathcal{A}(\mathcal{S}) \geq (1 - \delta)^2 (1 + \varepsilon')^{-1} \cdot \text{OPT}_{\text{MWISR}}. \quad (1)$$

Moreover, since the scaling happens around the anchors, the rectangles are still anchored in their associated input-points. It follows that, after possibly adding some trivial rectangles, the set \mathcal{S} is a solution for the ARP instance P . Note that for fixed δ and ε' , the total time spent to obtain \mathcal{S} is polynomial in n . The bottleneck here is the computation of \mathcal{S}' from the $|\mathcal{C}| \leq 2n^2 \cdot N^2$ candidate rectangles. By [1, Theorem 1], this can be done in time $n^{(\delta\varepsilon')^{-O(1/\varepsilon')}}$.

We now bound the area OPT_{ARP} of an optimal (α, β) -ARP solution $\mathcal{S}_{\text{ARP}}^*$ in terms of $\text{OPT}_{\text{MWISR}}$. To this end, fix a rectangle $R_p^* \in \mathcal{S}_{\text{ARP}}^*$ with anchor point $p \in P$. This rectangle is contained in at least one maximal candidate rectangle R_{max} . Let $i, j \in \{0, 1, \dots, N - 1\}$ be maximal such that $R_p^* \subseteq R_{\text{max}}^{i,j}$ and note that $\mathcal{A}(R_p^*) \leq \mathcal{A}(R_{\text{max}}) \cdot (1 - \delta)^{i+j}$. We say that R_p^* is *negligible* if $i = N - 1$ or $j = N - 1$. For each non-negligible $R_p^* \in \mathcal{S}_{\text{ARP}}^*$, define $R_p := R_{\text{max}}^{i+1, j+1}$ and denote by \mathcal{S}'' the set of all such rectangles. We consider the contribution of non-negligible and negligible rectangles to OPT_{ARP} separately.

- By construction, the contribution of non-negligible rectangles is at most $(1 - \delta)^{-2} \cdot \mathcal{A}(\mathcal{S}'')$. Furthermore, since $\mathcal{S}'' \subseteq \mathcal{C}$ and as the rectangles in \mathcal{S}'' are pairwise non-overlapping (as a shrunken subset of an ARP solution), \mathcal{S}'' is a solution to the MWISR instance \mathcal{C} . This implies that $\mathcal{A}(\mathcal{S}'') \leq \text{OPT}_{\text{MWISR}}$, which bounds the contribution of non-negligible rectangles to OPT_{ARP} by $(1 - \delta)^{-2} \cdot \text{OPT}_{\text{MWISR}}$.
- To bound the contribution of negligible rectangles, fix a negligible $R_p^* \in \mathcal{S}_{\text{ARP}}^*$ and note that $\mathcal{A}(R_p^*) \leq \mathcal{A}(R_{\text{max}}) \cdot (1 - \delta)^{N-1} \leq \text{OPT}_{\text{MWISR}} \cdot (1 - \delta)^{N-1} \leq \text{OPT}_{\text{MWISR}} \cdot \delta/n$, where the penultimate inequality holds since $\{R_{\text{max}}\}$ is a valid MWISR solution. Since there are at most n negligible rectangles in $\mathcal{S}_{\text{ARP}}^*$, they contribute at most $\delta \cdot \text{OPT}_{\text{MWISR}}$. Combined, we get that $\text{OPT}_{\text{ARP}} \leq (\delta + (1 - \delta)^{-2}) \cdot \text{OPT}_{\text{MWISR}}$. Using Equation (1), this implies

$$\mathcal{A}(\mathcal{S}) \geq (1 - \delta)^2 (1 + \varepsilon')^{-1} \cdot (\delta + (1 - \delta)^{-2})^{-1} \cdot \text{OPT}_{\text{ARP}} = (1 + \varepsilon)^{-1} \cdot \text{OPT}_{\text{ARP}}, \quad (2)$$

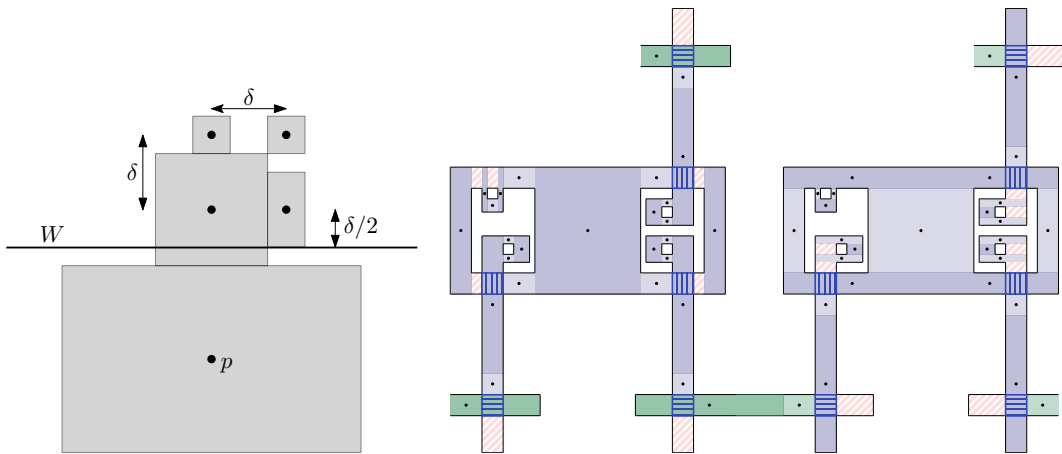
where the equality holds for small enough δ and appropriately chosen ε' . ◀

4 CARP is NP-Complete

In the decision variant of the center-anchored rectangle packing problem, we are given a value $A > 0$ and a finite point set $P \subseteq U$. The goal is to decide whether there is a valid rectangle set \mathcal{S} for P that covers an area of at least A . In this section we prove that this problem is NP-complete.

A useful observation is that if the input set contains four points in the corners of a tiny square with side length $\delta > 0$, then these points restrict the expansion of other points' rectangles but cannot expand their own rectangles beyond side length 2δ (see Figure 2a). This enables us to build walls that encode the graph structure of a suitable variant of maximum independent set (MIS).

► **Theorem 2.** *The center-anchored rectangle packing problem is NP-complete.*



(a) Building a wall W to the north of point p by adding four wall points. (b) Two node gadgets directly connected via a path gadget. The left node gadget is contracted, the right one expanded.

■ **Figure 2** Illustrations of the wall construction and a normalized solution for a pair of node gadgets.

The problem is easily seen to be in NP. Thus, the main challenge in proving Theorem 2 lies in the aforementioned reduction. Since the gadgets used in the construction as well as their interplay are somewhat involved, we give a high-level overview in Section 4.1. In Section 4.2, we describe in more detail how a suitable CARP instance is constructed from a given MIS instance. The analysis of this construction is given in Section 4.3.

4.1 Overview of the Reduction

The reduction is from the problem maximum independent set in 3-regular planar graphs (MIS3P), which is known to be NP-hard [12]. Given a MIS3P instance G , we use a result by Tamassia [18] to construct an orthogonal embedding of G . We then replace each node of G by a *node gadget* and each edge by a series of *path gadgets*. The number of path gadgets per edge corresponds (roughly) to the number of turns that it takes in the orthogonal embedding.

Gadgets & Rectangle Sets. Figure 2b illustrates the gadgets via a simple example where two node gadgets are directly² connected by a path gadget. Black dots represent the normal input points used to form center-anchored rectangles. Black lines represent walls built by placing four δ -spaced *wall points* at the orthogonal projections of each normal input point onto its closest wall in each direction. The horizontally striped blue areas connecting node gadget and path gadget are called *conflict areas*. The vertically striped blue areas at the entrance to the inner parts of the node gadget are called *compensation areas*. The shaded areas represent a valid example rectangle set for the (non-wall) input points. Diagonally striped red areas remain uncovered in the depicted solution.

Note that the rectangle sets of the two node gadgets in the depicted solution differ from one another. If, in a given solution, the points of a node gadget span rectangles exactly as in the right part of Figure 2b we say the node gadget is *expanded*. In particular, an expanded node gadget covers all three adjacent conflict areas. In the left part of Figure 2b, we see

² In our actual constructions, node gadgets are always connected via a chain of several path gadgets. Figure 2b neglects this technical detail in order to keep the illustration simple and instructive.

one possible example of a so called *contracted* node gadget. Such contracted node gadgets neither cover a neighboring conflict area (which is then covered by the adjacent path gadget), nor the neighboring compensation area (which is then not covered at all). A central part of our construction is that the latter case (where we loose an area of size b^2) happens only if the adjacent path gadget leads to another contracted gadget.

Normalized Solutions & Structural Properties. Our construction allows to prove the existence of a well-structured, almost optimal solution. We call a solution *normalized* if each wall point is assigned a $(\delta \times \delta)$ -square. In a normalized solution, non-wall points cannot expand their rectangles beyond adjacent walls. Additionally, we show the following properties.

- Lemma 4: There is a normalized solution approximating an optimal solution up to a factor of $1 - O(\delta/\varepsilon^2)$. Choosing δ suitably small allows us to consider normalized solutions only.
- Lemma 5: We call a solution normalized *with oriented paths* if the rectangles spanned by non-wall points are such that any series of connected path gadgets is alternatingly covered by “big” and “small” rectangles. An optimal normalized solution can be transformed into one with oriented paths without decreasing the covered area.
- Lemma 6: Given an optimal normalized solution with oriented paths, one can transform the rectangles spanned by non-wall points of node gadgets such that each node gadget is either expanded or contracted. This transformation changes only rectangles spanned by node gadgets and does not decrease the covered area.

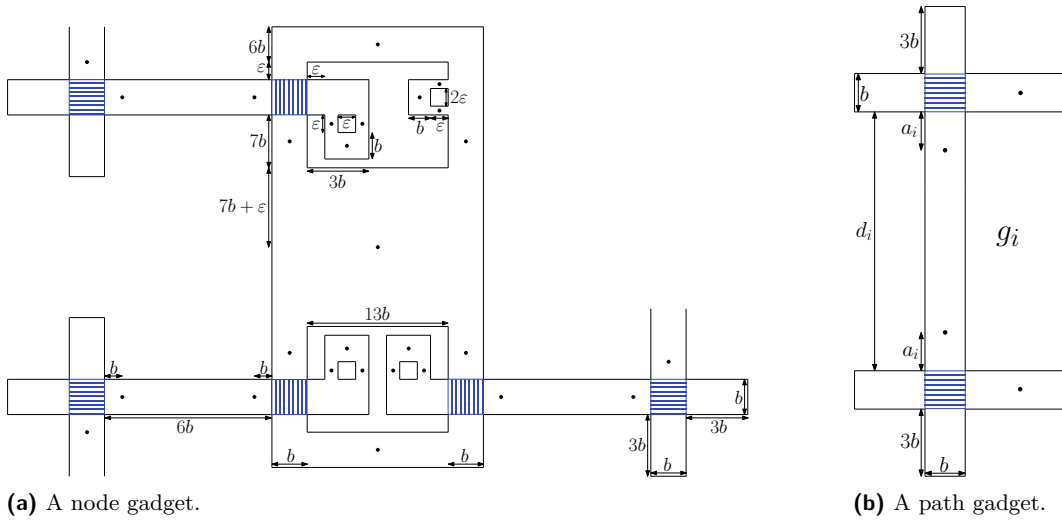
From this, we obtain a bijection between normalized solutions \mathcal{S} with oriented paths and only either expanded or contracted node gadgets and independent sets $I \subseteq V(G)$. Nodes $u \in I$ correspond to expanded node gadgets and nodes $u \notin I$ to contracted node gadgets.

Covered Area & Size of Independent Set. We derive a formula for the area covered by \mathcal{S} in terms of the size $|I|$ of the independent set. When a contracted node gadget does not cover one (or multiple) of its compensation areas, we charge these areas to the neighboring conflict area (which has the same size, b^2). See Figure 3 for an illustration of compensation and conflict areas. By this charging, the area covered by the inner part of any node gadget (whether expanded or contracted) differs only by the number of covered ε -stripes (small diagonally striped red areas within the node gadgets of size $O(\varepsilon \cdot b) \ll b^2$). For each path gadget, we always loose an area of size $3b^2$ at one of the two ends. Additionally, for each series of path gadgets connecting two contracted node gadgets, we loose an area b^2 of one of the conflict areas (using the charging).

Thus, ignoring negligible areas from the ε -stripes and wall-points, \mathcal{S} covers an area of size $\mathcal{A}(\mathcal{S}) = A_G + 3|I| \cdot b^2$. While A_G depends only on the graph G and its embedding, the second term stems from an independent set node being the only means of covering both critical areas of an incident edge. Thus, a solution covering the maximum area yields a maximum independent set and vice versa.

4.2 Construction of the CARP Instance

We start by formalizing the ideas of walls, depicted as black lines in our images. Fix a small $\delta > 0$ (whose value we specify later) and consider a point set P , a point $p \in P$, and a wall W (an axis aligned line segment), which is closest either in x - or y -direction. We modify P by adding four wall points that form an (axis aligned) square of side length δ on the (w.r.t. p) opposite side of the wall W . We do this in such a way that p and two of the wall points lie on a straight line and the wall point closer to p is at a distance of $\delta/2$ to W . See Figure 2a for an illustration.



■ **Figure 3** Detailed illustration of node and path gadgets with the respective side lengths. Conflict areas are horizontally striped. Compensation areas are vertically striped.

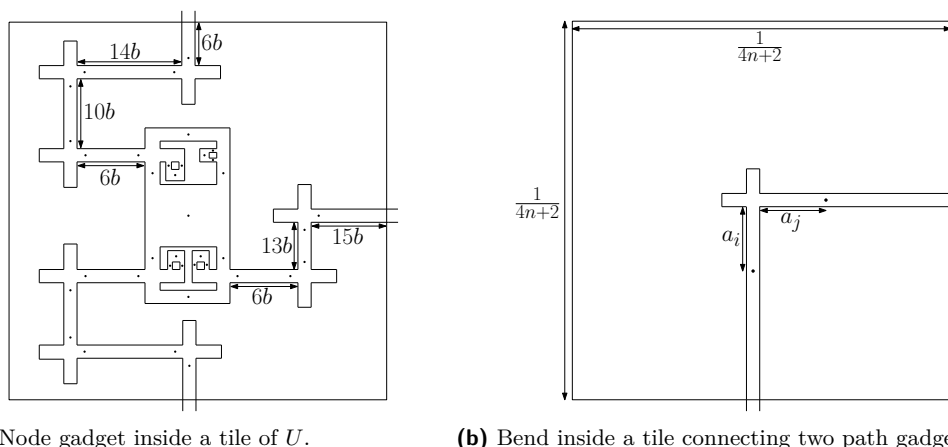
► **Observation 3.** Consider a valid solution for P after adding the four wall points for p . The rectangles spanned by p and by the wall points cannot transgress the wall by more than $\delta/2$. Moreover, the rectangles spanned by the four wall points cover an area³ of at most $4\delta^2$.

Gadgets. We now formally describe the node gadgets (representing nodes of the MIS3P instance) and the path gadgets (representing edges of the MIS3P instance). Figure 3a shows the details of a node gadget. Each input point is restricted by at most four walls. A node gadget has three outgoing paths each connected to a path gadget (corresponding to an incident edge). Figure 3b shows the details of a path gadget. Path gadgets are connected either to other path gadgets via a 90° turn (the crossings in the figure) or to a node gadget. There are two global parameters, the width of a path b and a small value $\varepsilon > 0$. We choose the former such that we can fit each gadget into a small *tile* whose size depends on the MIS3P instance, and we choose the latter small enough to ensure that certain non-coverable areas of node gadgets are negligible. A path gadget g_i has two additional parameters, $d_i > 0$ (roughly the length of the straight edge it represents) and $a_i := d_i/4 - b$ (specifying the exact placement of the non-wall points depending on d_i).

Orthogonal Embedding & Connecting the Gadgets. Next, consider an MIS3P instance G (i.e., a 3-regular planar graph) consisting of n nodes. We embed G in a $(4n + 2) \times (4n + 2)$ grid. This can be done in polynomial time (a simple application of a result from [18]). We then partition the unit square U into a grid of equal-sized tiles of width and height $1/(4n + 2)$. Depending on how the embedding behaves at a grid point $(i, j) \in \{1, \dots, 4n + 2\}^2$, we construct tile (i, j) from a suitable blueprint:

- *Grid point contains a node:* Use an appropriate rotation of the tile in Figure 4a.
- *Grid point contains an edge bend:* Use an appropriate rotation of the tile from Figure 4b.
- *Grid point contains part of an edge:* Insert tiles to form a straight path of width b , either from south to north or from east to west (determined by the orientation of the edge part).

³ While the rectangles may cover all of the square formed by the wall points, an area of size δ^2 , precisely $\frac{3}{4}$ of each rectangle's area lies outside this square. Thus, the total covered area is at most $\delta^2 + 3\delta^2$.

(a) Node gadget inside a tile of U .

(b) Bend inside a tile connecting two path gadgets.

■ **Figure 4** Two tiles used in our hardness construction. Further tiles include rotations of these as well as straight horizontal and vertical paths. Both sides of each tile have size $64b + 2\varepsilon$.

Note that outgoing path gadgets always leave a tile at the center of a tile boundary and that all parts are properly connected.

It remains to fix the parameters δ , b , and ε . These are chosen such that all parts of a node gadget tile (Figure 4a) fit exactly into the $1/(4n+2) \times 1/(4n+2)$ tiles and $\delta \ll \varepsilon \ll b$. The chosen lengths (Figure 3) guarantee that tile parts do not overlap. Our analysis also requires $2a_i \geq b$ (or, equivalently, $d_i \geq 6b$) for every path gadget g_i . One can easily verify that this is the case (see Section 4.2).

We now bound the size of the constructed CARP instance P_G for a graph G consisting of n nodes. Each tile of a node gadget contains 38 points. For each bend tile we add 2 points. Since there are n nodes and at most $3n+2$ bends [18, Lemma 7], the instance P contains at most $44n+4$ non-wall points. For each non-wall point we add at most 16 wall points. Thus, the size of P is bounded by $704n+64 = \Theta(n)$, implying that P_G can be constructed in polynomial time.

4.3 Analysis of the Constructed CARP Instance

Fix a MIS3P instance $G = (V, E)$ consisting of n nodes and let P_G denote the construction from the previous section. Let $W \subseteq P_G$ be the set of all wall points in P_G . Recall that a solution \mathcal{S} for the CARP instance P_G is called normalized if all points in W span $(\delta \times \delta)$ -rectangles. Note that in such a solution the rectangles spanned by points from $P_G \setminus W$ cannot overlap the walls. An *optimal normalized solution* is a normalized solution of maximum size. In the following, we state the key results of the analysis. Due to space constraints, we do not give the proofs in all detail here. For the complete proofs, we refer the reader to the full version of this paper.

Our first lemma shows that there exists a normalized solution that approximates an optimal solution up to a factor of $1 - O(\delta/\varepsilon^2)$.

► **Lemma 4.** *There exists a normalized solution for CARP on the instance P_G of area at least $(1 - 2\delta/\varepsilon^2) \cdot \text{OPT}_{\text{CARP}(P_G)}$.*

The next pair of lemmas form the center of our arguments. They show the existence of optimal normalized solutions exhibiting useful structural properties for both the path and node gadgets.

8:10 On the Complexity of Anchored Rectangle Packing

► **Lemma 5.** *There exists an optimal normalized solution in which the two rectangles spanned by a path gadget g_i 's two non-wall points have width (or, depending on orientation, height) b , and one of them has height (width) $2a_i$, the other one $2(a_i + 4b)$. In particular, the rectangles of each path gadget cover exactly one of the gadget's two conflict areas.*

- **Lemma 6.** *There exists an optimal normalized solution such that the following holds:*
- *Each node gadget is either contracted or expanded.*
 - *No two adjacent node gadgets are expanded.*
 - *For each pair of contracted adjacent node gadgets there is a (uniquely identified) uncovered area of size b^2 inside one of the node gadgets.*

The previous two lemmas hold simultaneously. NP-hardness follows from these results as described in Section 4.1.

5 Resource Augmentation

In this section, we investigate the lower-left-anchored rectangle packing problem. We study two different types of resource augmentation. In the first type, which we call *perturbation-augmentation*, each solution rectangle does not need to be anchored exactly in the corresponding point but instead its anchor may be up to an ε distance away from that point (see Figure 5, center). We define an ε -grid $\Gamma = (V, \mathcal{L})$ as a set of points $V = \{(x, y) \in U \mid x = \varepsilon \cdot k_x \wedge y = \varepsilon \cdot k_y \wedge k_x, k_y \in \mathbb{N}\}$ together with a family of grid-cells $\mathcal{L} = \{[x, x + \varepsilon] \times [y, y + \varepsilon] \subseteq U \mid (x, y) \in V\}$. The perturbation-augmentation allows us to focus on solutions where all vertices of rectangles are points on an ε -grid Γ , thereby allowing us to enumerate all possible sets of interior-disjoint, axis-aligned rectangles with vertices in V . We call such a solution a *grid-point solution*. We show that (i) there exists a polynomial-time algorithm that finds the optimal grid-point solution and (ii) the optimal grid-point solution covers at least as much area as an optimal (unrelaxed) solution.

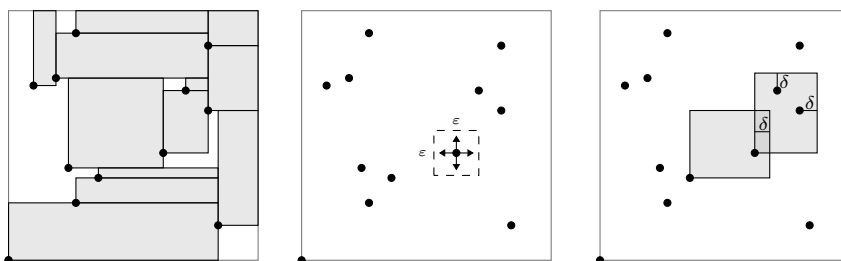
Before formally stating the theorem, we introduce some notation. Let $\text{dist}_\infty(x, y)$ denote the distance of two points $x, y \in \mathbb{R}^2$ in the ℓ_∞ -norm. For a set $X \subseteq \mathbb{R}^2$ and a point $y \in \mathbb{R}^2$, we write $\text{dist}_\infty(y, X) = \inf_{x \in X} \text{dist}_\infty(y, x)$ and $\overline{X} = \mathbb{R}^2 \setminus X$ for the set's complement. We say that a set of interior-disjoint rectangles $\{R_p\}_{p \in P}$ is ε -valid, if $\text{dist}_\infty(p, \ell(R_p)) < \varepsilon$ and $\text{dist}_\infty(p, \overline{R_p}) < \varepsilon$ for all $p \in P$, where $\ell(R_p)$ denotes the lower-left corner of R_p .

► **Theorem 7.** *For every $\varepsilon > 0$ and a finite point-set $P \subseteq U$, there exists a polynomial time algorithm that computes an ε -valid set of interior-disjoint rectangles that cover an area of at least $\text{OPT}_{\text{LLARP}(P)}$.*

The second type of resource augmentation we call *overlap-augmentation*. It relaxes the condition that the rectangles need to be disjoint. More specifically, each pair of rectangles is allowed to overlap by a thin strip of width at most δ . Note that this implies that a rectangle may also contain input points as long as they are no more than a δ -distance away from the boundary (see Figure 5, right).

Again, we require additional definitions. A set of rectangles $\{R_p\}_{p \in P}$ is called a δ -LLARP, if the rectangle R_p is lower-left-anchored in p and $\sup_{x \in R_{p'}} \text{dist}_\infty(x, \overline{R_p}) < \delta$, for all $p' \in P$. We show that we can transform an ε -valid grid-point solution into a δ -LLARP while only losing a small fraction of the covered area.⁴

⁴ Naturally, any area that is covered by multiple rectangles is counted only once.



■ **Figure 5** An optimal lower-left-anchored rectangle packing (left), and illustrations of the two kinds of resource augmentation that are used in the paper.

► **Theorem 8.** *For every $\varepsilon > 0$ and a finite point-set $P \subseteq U$, there exists a polynomial time algorithm that outputs an $\frac{\varepsilon}{11}$ -LLARP that covers an area of at least $(1-\varepsilon) \cdot \text{OPT}_{\text{LLARP}(P)}$.*

■ **Algorithm 1** Algorithm using resource augmentation of the perturbation-type.

```

1:  $\Gamma \leftarrow \varepsilon$ -grid in  $U$ ,  $\mathcal{H} \leftarrow \emptyset$ 
2: for every configuration  $C$  in  $\Gamma$  do
3:   if  $C$  is  $\varepsilon$ -valid for  $P$  then
4:     for every grid-point solution  $\mathcal{S} := \{R_q\}_{q \in C}$  of  $\text{LLARP}(C)$  do
5:        $\mathcal{H} \leftarrow \mathcal{H} \cup \{\mathcal{S}\}$ 
6: return  $\mathcal{S} \in \mathcal{H}$  maximizing  $\mathcal{A}(\mathcal{S})$ 

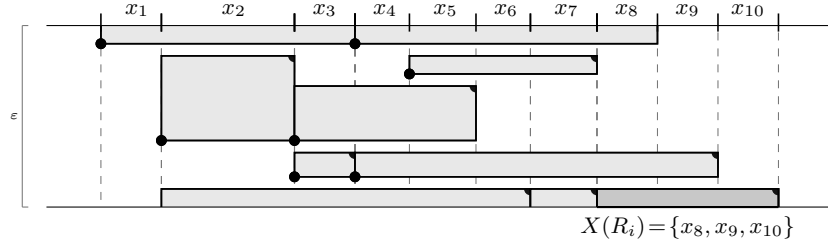
```

Proof of Theorem 7. Let $\Gamma = (V, \mathcal{L})$ be an ε -grid in U and denote by k the number of grid-cells. Without loss of generality, assume that all points of P lie in the interior of one of the $O(\varepsilon^{-2})$ grid-cells. A *grid configuration* on Γ is a subset of the grid points $C \subseteq V$. We say that a grid configuration C on Γ is ε -valid for P if there is a surjective mapping $\varphi : P \rightarrow C$, such that $\text{dist}_\infty(p, \varphi(p)) \leq \varepsilon$. Note that any LLARP for an ε -valid configuration C for P can be extended to an ε -valid solution for P by adding degenerate rectangles anchored at points in C .

Algorithm 1 enumerates all grid configurations on Γ that are ε -valid for P . Then, for each configuration C , it computes all grid-point solutions $\{R_q\}_{q \in C}$ of $\text{LLARP}(C)$. Among all enumerated solutions, we keep one that maximizes the covered area.

First, we show that the running time of this algorithm is polynomial. Note that any grid cell has four grid points as its vertices. Since any point $p \in P$ lies in the interior of some grid cell $L \in \mathcal{L}$, an ε -valid grid configuration must contain at least one of the four vertices of L . Thus, we can construct any ε -valid grid configuration by choosing for each grid cell $L \in \mathcal{L}$ and each input point $p \in P \cap L$, a vertex of L as its image in the mapping φ . However, since whenever multiple rectangles are anchored in the same point at most one can be non-degenerate, in each $L \in \mathcal{L}$ it suffices to decide on one of at most 15 cell configurations $L \cap C$ using either 0, 1, 2, 3, or 4 of the grid cell's vertices. In particular, any number of contained input points larger than 3, yields the same set of possible cell configurations. Thus, by counting the number of input points contained in each grid cell (in time $O(n)$) and then enumerating at most 15^k grid configurations, we obtain all ε -valid grid configurations. For each such configuration C , we obtain all corresponding ε -valid grid-point solutions by picking for each $q \in C$ one of the $O(\varepsilon^{-2})$ grid points above and to the right of q as the upper-right corner of the rectangle and checking interior disjointness. This amounts to at most $O(\varepsilon^{-2\varepsilon^{-2}})$ solutions per grid configuration. Thus, in total, the running time of Algorithm 1 is linear in n and doubly exponential in $\frac{1}{\varepsilon}$.

8:12 On the Complexity of Anchored Rectangle Packing



■ **Figure 6** A row B of the ε -grid F and the corresponding rectangles of \mathcal{S}_B^* .

It remains to show that the computed grid-point solution covers at least as much area as an optimal lower-left-anchored rectangle packing $\mathcal{S}_{\text{LLARP}(P)}^* = \{R_1, R_2, \dots, R_n\}$ without resource augmentation. We transform $\mathcal{S}_{\text{LLARP}(P)}^*$ to a grid-point solution by “snapping” the corners of its rectangles to Γ . We do this by first snapping the lower-left and upper-right corners of each rectangle to either of the horizontal grid-lines directly above and below the corner in question. Afterwards, the same is done analogously for the vertical grid-lines, which we omit in our discussion.

Consider one row of the grid, say $B = [0, 1] \times (y, y + \varepsilon)$, for $(x, y) \in V$ (see Figure 6). We describe how the lower-left and upper-right corners within this row can be snapped up or down without reducing the total area. Partition the row horizontally into segments x_1, x_2, \dots, x_r in-between the x -coordinates of corners of the rectangles $\mathcal{S}_B^* := \{R_i\}_{i=1}^m \subseteq \mathcal{S}_{\text{LLARP}(P)}^*$ that have a corner in the row. We associate a variable $h_i \in [0, 1]$ with each rectangle R_i denoting the height of $R_i \cap B$ relative to the height of B . Snapping the rectangles in this row can be seen as rounding each such variable h_i either up to 1 or down to 0 (i.e., expanding the rectangle to span the full row height or collapsing it such that it does not appear in the row anymore). We aim to do this in a manner, which neither introduces overlaps nor decreases the covered area. This snapping problem is represented by the following integer linear program.

$$\max \sum_{i=1, \dots, m} h_i \cdot w(R_i) \quad (3)$$

$$\text{s.t.} \quad \sum_{i: x_j \in X(R_i)} h_i \leq 1 \quad \text{for } j = 1, \dots, r \quad (4)$$

$$h_i \in \{0, 1\} \quad \text{for } i = 1, \dots, m. \quad (5)$$

Here, $w(R_i)$ denotes the width of rectangle R_i , and the set $X(R_i)$ consists of all segments x_i intersecting R_i . The objective function (3) maximizes the covered area while constraints in (4) ensure that (in an integer solution) rectangles do not overlap. The binary constraints in (5) force the rectangles to either span the entire height or nothing of the row. We note that the linear program is inspired by a similar one for the demand flow problem [10].

It is easy to verify that \mathcal{S}_B^* corresponds to a feasible solution for the linear programming relaxation of (3)–(5) and thus lower bounds the optimal value of the LP-relaxation. Since the sets $X(R_i)$ contain consecutive line segments, the LP satisfies the consecutive ones property [17]. Therefore, constraints (4) yield a totally unimodular matrix and the LP is integral [17]. It follows, that there is an integral solution that solves the relaxed linear program optimally and induces a partial snapping of area at least $\text{OPT}_{\text{LLARP}(P)}$. Repeating this argument for all rows and columns of Γ yields a grid-point solution with an area of at least $\text{OPT}_{\text{LLARP}(P)}$. ◀

■ **Algorithm 2** Algorithm using resource augmentation of the overlap-type.

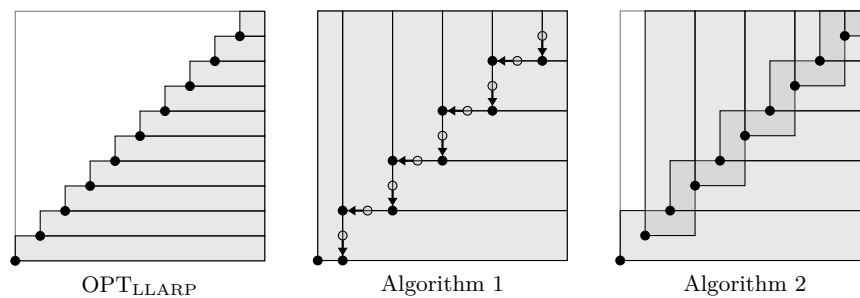
-
- 1: Compute $S_I = \text{Algorithm 1}(P, \frac{\varepsilon}{22})$
 - 2: **for** all $R_p \in S_I$ **do**
 - 3: $R'_p \leftarrow$ the rectangle spanned by p and $u(R_p) + (\frac{\varepsilon}{22}, \frac{\varepsilon}{22})$
 - 4: **if** $u(R'_p) \notin U$ **then**
 - 5: $u(R'_p) \leftarrow \arg \min_{x \in U} \|x - u(R'_p)\|$
 - 6: **return** $S_{II} = \{R'_p\}_{p \in P}$
-

Proof of Theorem 8. We show that Algorithm 2 satisfies the conditions of Theorem 8, that is it runs in polynomial time and outputs a $\frac{\varepsilon}{11}$ -LLARP of area at least $(1 - \varepsilon) \cdot \text{OPT}_{\text{LLARP}}(P)$. In comparison to the previous case, input points cannot be moved, but the rectangles may somewhat overlap.

Consider the $\frac{\varepsilon}{22}$ -valid solution $S_I = \{R_p\}_{p \in P}$ of Algorithm 1 with input $(P, \frac{\varepsilon}{22})$. In constructing S_I , we move the input-points $p \in P$ to grid-points $q_p \in F$ spanning R_p , the upper-right corner of which we denote by $u(R_p)$. We transform the solution of Algorithm 1 to obtain an $\frac{\varepsilon}{11}$ -LLARP.

Shift each $u(R_p)$ up and to the right by $\frac{\varepsilon}{22}$ to obtain $u(R_p)'$. The transformed solution S'_I consists of the rectangles uniquely defined by the lower-left corners $p \in P$ and their corresponding upper-right corners $u(R_p)'$. Since $\text{dist}_\infty(p, q_p) < \frac{\varepsilon}{22}$, moving $u(R_p)$ by $(\frac{\varepsilon}{22}, \frac{\varepsilon}{22})$ ensured that transforming the solution did not decrease its size. The overlap of rectangles is bounded by $2 \cdot \frac{\varepsilon}{22} = \frac{\varepsilon}{11}$.

However, moving the solution may cause some rectangles to protrude from the unit square. We prune this excess area, by moving the respective upper-right corners back into the square to obtain our final solution S_{II} . Due to this pruning, we lose an area of size at most $2 \cdot \frac{\varepsilon}{22} = \frac{\varepsilon}{11} \leq \varepsilon \cdot \text{OPT}_{\text{LLARP}}(P)$, where the inequality follows from the lower bound of $\frac{1}{11}$ on $\text{OPT}_{\text{LLARP}}(P)$ [11]. This implies, that $\mathcal{A}(S_{II}) \geq (1 - \varepsilon) \cdot \text{OPT}_{\text{LLARP}}(P)$. ◀



■ **Figure 7** Algorithms Algorithm 1 and Algorithm 2 applied to a simple instance.

References

- 1 Anna Adamaszek, Parinya Chalermsook, and Andreas Wiese. How to Tame Rectangles: Solving Independent Set and Coloring of Rectangles via Shrinking. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*, volume 40 of *LIPIcs*, pages 43–60. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.43.

- 2 Anna Adamaszek and Andreas Wiese. Approximation Schemes for Maximum Weight Independent Set of Rectangles. In *54th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2013)*, pages 400–409. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.50.
- 3 Anna Adamaszek and Andreas Wiese. A quasi-PTAS for the Two-Dimensional Geometric Knapsack Problem. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 1491–1505. SIAM, 2015. doi:10.1137/1.9781611973730.98.
- 4 Hugo A. Akitaya, Matthew D. Jones, David Stalf, and Csaba D. Tóth. Maximum Area Axis-Aligned Square Packings. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117 of *LIPICs*, pages 77:1–77:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPICs.MFCS.2018.77.
- 5 Kevin Balas, Adrian Dumitrescu, and Csaba D. Tóth. Anchored rectangle and square packings. *Discrete Optimization*, 26:131–162, 2017. doi:10.1016/j.disopt.2017.08.003.
- 6 Nikhil Bansal and Arindam Khan. Improved Approximation Algorithm for Two-Dimensional Bin Packing. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 13–25. SIAM, 2014. doi:10.1137/1.9781611973402.2.
- 7 Sergey Bereg, Adrian Dumitrescu, and Minghui Jiang. Maximum Area Independent Sets in Disk Intersection Graphs. *Int. J. Comput. Geometry Appl.*, 20(2):105–118, 2010. doi:10.1142/S0218195910003220.
- 8 Sergey Bereg, Adrian Dumitrescu, and Minghui Jiang. On Covering Problems of Rado. *Algorithmica*, 57(3):538–561, 2010. doi:10.1007/s00453-009-9298-z.
- 9 Therese C. Biedl, Ahmad Biniiaz, Anil Maheshwari, and Saeed Mehrabi. Packing Boundary-Anchored Rectangles. In *Proceedings of the 29th Canadian Conference on Computational Geometry (CCCG 2017)*, pages 138–143, 2017.
- 10 Chandra Chekuri, Marcelo Mydlarz, and F. Bruce Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Trans. Algorithms*, 3(3):27, 2007. doi:10.1145/1273340.1273343.
- 11 Adrian Dumitrescu and Csaba D. Tóth. Packing anchored rectangles. *Combinatorica*, 35(1):39–61, 2015. doi:10.1007/s00493-015-3006-1.
- 12 Bojan Mohar. Face Covers and the Genus Problem for Apex Graphs. *J. Comb. Theory, Ser. B*, 82(1):102–117, 2001. doi:10.1006/jctb.2000.2026.
- 13 Richard Rado. Some covering theorems (I). *Proc. London Math. Soc.*, 51:232–264, 1949.
- 14 Richard Rado. Some covering theorems (II). *Proc. London Math. Soc.*, 53:243–267, 1951.
- 15 Richard Rado. Some covering theorems (III). *Proc. London Math. Soc.*, 42:127–130, 1968.
- 16 Tibor Radó. Sur un problème relatif à un théorème de Vitali. *Fund. Math.*, 11:228–229, 1928.
- 17 Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.
- 18 Roberto Tamassia. On Embedding a Graph in the Grid with the Minimum Number of Bends. *SIAM J. Comput.*, 16(3):421–444, 1987. doi:10.1137/0216030.
- 19 William Thomas Tutte, editor. *Recent Progress in Combinatorics: Proceedings of the 3rd Waterloo Conference on Combinatorics*. Academic Press, 1969.
- 20 Peter Winkler. Packing Rectangles. *Mathematical Mind-Benders*, pages 133–134, 2007.

Quantum Walk Sampling by Growing Seed Sets

Simon Apers

Inria, Paris, France

CWI, Amsterdam, The Netherlands

simon.apers@inria.fr

Abstract

This work describes a new algorithm for creating a superposition over the edge set of a graph, encoding a quantum sample of the random walk stationary distribution. The algorithm requires a number of quantum walk steps scaling as $\tilde{O}(m^{1/3}\delta^{-1/3})$, with m the number of edges and δ the random walk spectral gap. This improves on existing strategies by initially growing a classical seed set in the graph, from which a quantum walk is then run.

The algorithm leads to a number of improvements: (i) it provides a new bound on the setup cost of quantum walk search algorithms, (ii) it yields a new algorithm for st -connectivity, and (iii) it allows to create a superposition over the isomorphisms of an n -node graph in time $\tilde{O}(2^{n/3})$, surpassing the $\Omega(2^{n/2})$ barrier set by index erasure.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Graph algorithms analysis

Keywords and phrases Quantum algorithms, Quantum walks, Connectivity, Graph theory

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.9

Acknowledgements This work benefited from discussions with Alain Sarlette, Stacey Jeffery, Anthony Leverrier, Ronald de Wolf, André Chailloux and Frédéric Magniez. Part of this work was supported by the CWI-Inria International Lab.

1 Introduction and Summary

Sampling from the stationary distribution of a random walk is a common and valuable tool in the design of algorithms [32]. It underlies the Markov chain Monte Carlo paradigm, and plays a central role in a wide range of approximation algorithms for graph problems. In this work we investigate the quantum counterpart of this task - generating quantum samples from the random walk stationary distribution. Given query access to some graph $G = (\mathcal{V}, \mathcal{E})$ with m edges, we wish to create the quantum state

$$|\pi\rangle = \frac{1}{\sqrt{m}} \sum_{(i,j) \in \mathcal{E}} |i, j\rangle, \quad (1)$$

which is a superposition over the edges of the graph. Measuring the first register of this state, and discarding the second register, indeed returns the random walk stationary distribution. Creating such a quantum sample of a classical stationary distribution forms a crucial primitive for a range of algorithms: the so-called “setup cost” in quantum walk search algorithms [28, 25] refers to the cost of generating a state such as $|\pi\rangle$, quantum algorithms for speeding up MCMC [2, 33, 39, 30] build on the possibility of efficiently creating quantum samples, and a number of quantum algorithms for solving graph problems [38, 23] require the generation of a superposition over the edges of a graph.

We develop a new quantum algorithm for creating the quantum sample (1), given only local query access to the graph. Our algorithm improves the query and time complexity of the folklore approach to quantum sampling from $\tilde{O}(m^{1/2}\delta^{-1/2})$ to $\tilde{O}(m^{1/3}\delta^{-1/3})$. We do so by growing a classical seed set from the initial node. This incurs a payoff in the space complexity, increasing it from $\tilde{O}(1)$ to $\tilde{O}(m^{1/3}\delta^{-1/3})$. As a demonstration of our algorithm,



© Simon Apers;

licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 9; pp. 9:1–9:12

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

we discuss a new approach to solving st -connectivity: generate a superposition over the connected components of s and t , and compare these states. This approach generalizes the notorious quantum state generation strategy for solving graph isomorphism. Concerning the latter, we show that our algorithm allows to create a superposition over the isomorphisms of a given n -vertex input graph in $\tilde{O}(2^{n/3})$ steps. This surpasses the $\Omega(2^{n/2})$ index erasure barrier by Ambainis et al [6]. In a similar way we can create a superposition over the elements of a black box group in $\tilde{O}(2^{n/3})$ steps, where 2^n is the number of group elements.

1.1 Query Model

We assume throughout this work that we only have “local” query access to the graph $G = (\mathcal{V}, \mathcal{E})$: we are given an initial node $j \in \mathcal{V}$, and we can query for its degree and neighbors. Such queries fall under the so-called *adjacency array model* [20] or *bounded degree model* [21] (although we do not assume the degree is bounded), which is very natural when studying random walk algorithms. However, departing from these models, and justifying the term “local”, we will not assume direct access to or prior knowledge about \mathcal{V} , apart from the initial node. For comparison, in [20] the node set \mathcal{V} is given as a list, and in [21] access to uniformly random nodes is assumed. In this sense our work is more in line with graph exploration algorithms as considered e.g. in [34], or more recently in [17].

Since our algorithm strongly builds on the use of quantum walks, we will alternatively express the complexity of our results as a function of the number of quantum walk steps. Also in such case the denominator “local” query access is justified, since a single quantum walk step from a certain node only accesses the neighbors of that node.

1.2 Quantum Walk Sampling Algorithm

Our algorithm builds on the folklore approach to creating $|\pi\rangle$, discussed in e.g. [31, 39, 30, 29]. Starting from some initial state $|j\rangle$ localized on a node $j \in \mathcal{V}$, this approach combines quantum phase estimation and amplitude amplification on the quantum walk operator associated to the graph. We detail this scheme in Section 2.2. The scheme requires $\tilde{O}(m^{1/2}\delta^{-1/2})$ QW steps on the graph, where δ is the random walk spectral gap, and the factor $m^{1/2}$ stems from the small projection of the initial state onto $|\pi\rangle$.

In the present work we improve on this scheme by initially doing some “classical work”: we first use classical means to grow a seed set around the initial vertex. Briefly ignoring the δ -dependency, we grow the set to have size $\Theta(m^{1/3})$. We can then use a special data structure to generate and reflect around a quantum superposition over this set, which now has a $\Omega(m^{-1/3})$ overlap with the target state. Reinvoking the folklore scheme from this state then allows to retrieve $|\pi\rangle$, now only requiring $\tilde{O}(m^{1/3})$ queries. This approach leads to the following result.

► **Theorem 1.** *Given a lower bound $\gamma \leq \delta$ on the spectral gap, it is possible to create the quantum state $|\pi\rangle$ using $\tilde{O}(m^{1/3}\gamma^{-1/3})$ time, space and QW steps.*

Apart from the log-factors, the combined dependency on m and δ is optimal. Indeed it is tight on e.g. the cycle graph, which has $m = n$ and $\delta = n^{-2}$, giving an $\tilde{O}(n)$ steps algorithm. Since the diameter of the cycle is $\Omega(n)$, this is optimal when assuming local query access. We also note that, if in addition we are given a bound $D \geq d_M$ on the maximum degree (in e.g. the array model this is always given), then we can implement our algorithm using $\tilde{O}(m^{1/3}\gamma^{-1/3}D^{1/3})$ degree and neighbor queries.

The algorithm gives a direct bound on the so-called setup cost of quantum walk search algorithms in the MNRS framework [28] as a function of the update cost (i.e., the cost of implementing a quantum walk step). The increased space complexity of our algorithm, $\tilde{O}(m^{1/3}\delta^{-1/3})$ as compared to $\tilde{O}(1)$ for the folklore approach, is very similar to the payoff in space versus time or query complexity in the collision finding algorithm of Brassard et al [15] and the element distinctness algorithm of Ambainis [4].

1.3 Application to st -connectivity

Our QW sampling algorithm yields a new approach for solving st -connectivity, somewhat similar to the approach taken by Watrous in [38]: generate a superposition over the edges in the connected components of s resp. t , and compare the resulting states. As we prove in Proposition 9, this requires $\tilde{O}(m^{1/3}\gamma^{-1/3})$ QW steps, where γ is a lower bound on the spectral gaps of the connected components of s and t . Our algorithm outperforms the existing quantum algorithms for st -connectivity [20, 13, 12, 23] on for instance sparse graphs with a good spectral gap.

The approach generalizes a well-known strategy to solving graph isomorphism on a quantum computer [2] (called “component mixing” in [27]): generate superpositions over the isomorphisms of each graph, and compare the resulting states. In [6], Ambainis et al aimed to prove a lower bound on this approach by abstracting it to the so-called index erasure problem. For this generalized problem, they prove a lower bound of $\Omega(2^{n/2})$. They argue that the same bound holds for creating a superposition over graph isomorphisms, be it under the condition that the algorithm makes no use of the structure of the problem. We show that, by exploiting the structure of the problem, we can indeed use our quantum walk sampling algorithm to surpass this bound. Thereto we consider the graph whose node set consists of isomorphisms of the input graph, and whose edge set arises from performing pairwise transpositions on the nodes (i.e., on the adjacency matrices of the isomorphisms). Using our quantum walk sampling algorithm on this graph yields the following corollary.

► **Corollary 2.** *Given an n -node input graph g , it is possible to create a superposition over the isomorphisms of g in $\tilde{O}(2^{n/3})$ steps.*

Completing the associated st -connectivity algorithm, we find an $\tilde{O}(2^{n/3})$ quantum algorithm for graph isomorphism. Using the existing quantum algorithms for st -connectivity, this approach would require $\Omega(2^{n/2})$ steps. Clearly the improved performance still falls terribly short of current (classical) algorithms for graph isomorphism, most notably the quasi-polynomial algorithm by Babai [10], yet it provides a clear demonstration of how the readily accessible structure of the problem allows to surpass the index erasure bound.

A similar strategy exists for solving the group non-membership problem on a quantum computer, as proposed by Watrous [37], requiring to generate a superposition over the elements of a finite black box group. Using the random walk algorithm by Babai [9] for generating uniformly random group elements, we can similarly generate this superposition in $\tilde{O}(2^{n/3})$ steps, when 2^n is the number of group elements.

1.4 Open Questions

This work leaves open a number of questions and possible applications, some of which we summarize below:

- *Quantum sampling for general Markov chains or stoquastic Hamiltonians.* In this work we only consider the quantum sampling problem for random walks. Generalizing our approach to more general Markov chains could lead to improvements on quantum MCMC

algorithms [2, 33], or the preparation of many body ground states [30] and Gibbs states [36]. The main bottleneck to such generalization seems to be the classical construction of seed sets which have an appropriate overlap with the goal quantum state. Even more generally, one could consider the preparation of ground states of Hamiltonians. For e.g. the special case of stoquastic Hamiltonians, which are known to have a nonnegative ground state, it should be possible to construct a seed set with improved overlap with the ground state.

- *Faster quantum fast-forwarding.* In former work by the author [8], a quantum algorithm was proposed for quantum sampling a t -step Markov chain. If the Markov chain has transition matrix P , and is started from a node i , the algorithm has complexity $\tilde{O}(\|P^t |i\rangle\|^{-1} t^{1/2}) \in \tilde{O}(m^{1/2} t^{1/2})$. Using ideas from the present work, it seems very feasible that we can improve this complexity to $\tilde{O}(\|P^t |i\rangle\|^{-2/3} t^{1/2}) \in \tilde{O}(m^{1/3} t^{1/2})$. Rather than using a breadth-first search to grow the seed set, as in the present work, it seems more suitable to use random walk techniques as in [34, 7]. As a byproduct, this would yield an improved quantum expansion tester, combining the speedups of [5] and [8].
- *Quantum search in \sqrt{HT} .* Our algorithm does not suffer from the so-called “symmetry barrier” in quantum algorithms: we can go from $|j\rangle$ to $|\pi\rangle$ more easily than from $|\pi\rangle$ to $|j\rangle$. Indeed, if for instance the underlying graph is an expander, then the former takes $O(n^{1/3})$ queries, whereas the latter takes $\Omega(n^{1/2})$ queries by the search lower bound.

An open problem related to this is the following: given an initial node s in a graph, can we find a node t in $O(HT_{s,t}^{1/2})$ QW steps, with $HT_{s,t}$ the hitting time from s to t ? Currently the best algorithm for this problem is by Belovs [12], which solves it in $O(CT_{s,t}^{1/2})$, with $CT_{s,t} = H_{s,t} + H_{t,s}$ the commute time between s and t . Since the commute time is symmetric between s and t , this obeys the aforementioned symmetry barrier. However, the commute time can be much larger than the hitting time from s to t , hence the open question of whether we can improve this performance to $O(HT_{s,t}^{1/2})$, thereby necessarily breaking this symmetry e.g. by using our techniques.

1.5 Outline

In Section 2 we discuss the graph and query model (Section 2.1), and provide the necessarily preliminaries on random walks and quantum walks (Section 2.2). In Section 3 we propose an algorithm for growing a classical seed set (Section 3.1), we discuss the data structure (Section 3.2), and we propose our QW sampling algorithm (Section 3.3). Finally in Section 4 we discuss the application of our QW sampling algorithm for solving st -connectivity (Section 4.1), and we demonstrate it for the special case of graph isomorphism testing (Section 4.2).

2 Preliminaries: Queries and Walks

2.1 Graph and Query Model

Throughout the paper we assume local query access to an undirected graph $G = (\mathcal{V}, \mathcal{E})$, with \mathcal{E} a subset of the ordered pairs $\mathcal{V} \times \mathcal{V}$, such that $(i, j) \in \mathcal{E} \Leftrightarrow (j, i) \in \mathcal{E}$. We denote $|\mathcal{V}| = n$ and $|\mathcal{E}| = m$. For any $\mathcal{S} \subseteq \mathcal{V}$, we let $\mathcal{E}(\mathcal{S})$ denote the set of edges starting in \mathcal{S} , i.e.,

$$\mathcal{E}(\mathcal{S}) = \{(i, j) \in \mathcal{E} \mid i \in \mathcal{S}\}.$$

For any $i \in \mathcal{V}$, we let $d(i) = |\mathcal{E}(\{i\})|$ denote the degree of i , the maximum degree $d_M = \max_{i \in \mathcal{V}} d(i)$, and $d(\mathcal{S}) = |\mathcal{E}(\mathcal{S})| = \sum_{i \in \mathcal{S}} d(i)$ denotes the total degree of a set $\mathcal{S} \subseteq \mathcal{V}$. A single query consists of either of the following:

- *degree query:* given $i \in \mathcal{V}$, return degree $d(i)$
- *neighbor query:* given $i \in \mathcal{V}$, $k \in [d(i)]$, return k -th neighbor of i

As an alternative query model we will also consider the quantum walk model, or so-called MNRS framework, as proposed in [28] in the context of quantum walk search. The model associates abstract costs to different operations¹:

- *setup cost*: the cost of preparing the quantum sample $|\pi\rangle = m^{-1/2} \sum_{(i,j) \in \mathcal{E}} |i,j\rangle$
- *update cost*: the cost of implementing a quantum walk step. See Section 2.2 for details.

For search problems an additional *checking cost* is considered, yet this will not be relevant here. In [16] it is proven that the update cost or quantum walk step for a node i can be simulated using $O(d(i)^{1/2})$ degree and neighbor queries. From our work it follows that the setup cost can be simulated using $\tilde{O}(m^{1/3}\delta^{-1/3})$ QW steps, or $\tilde{O}(m^{1/3}d_M^{1/3}\delta^{-1/3})$ degree and neighbor queries.

2.2 Random Walks and Quantum Walks

From some initial seed vertex $j \in \mathcal{V}$, we can use degree and neighbor queries to implement a random walk over \mathcal{V} . The transition matrix P describing such a walk is defined by $P(i,j) = 1/d(i)$ if $(i,j) \in \mathcal{E}$, and $P(i,j) = 0$ elsewhere. If the graph is connected and nonbipartite, then the random walk converges to its stationary distribution π , defined by $\pi(i) = d(i)/m$ for any $i \in \mathcal{V}$. If we order the eigenvalues of P (with multiplicities) as $1 = \lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n \geq -1$, then the rate at which the walk converges to π is bounded by the spectral gap $\delta = 1 - \max\{|\lambda_2|, |\lambda_n|\}$ [26].

Quantum walks (QWs) form an elegant quantum counterpart to random walks on graphs. Following the exposition in [28], they are naturally defined over a vector space associated to the edge set

$$\mathcal{H}_{\mathcal{E}} = \text{span}_{\mathbb{C}}\{|i,j\rangle \mid (i,j) \in \mathcal{E}\}.$$

A quantum walk over $\mathcal{H}_{\mathcal{E}}$ is now defined as the unitary operator $W = SR_{\mathcal{E}}$, where $R_{\mathcal{E}}$ is a reflection around the subspace $\text{span}_{\mathbb{C}}\{|\psi_i\rangle \mid i \in \mathcal{V}\}$, with

$$|\psi_i\rangle = \frac{1}{\sqrt{d(i)}} \sum_{(i,j) \in \mathcal{E}} |i,j\rangle,$$

and S represents the swap operator defined by $S|i,j\rangle = |j,i\rangle$. The cost of implementing the QW operator W is called the update cost, but can alternatively be implemented using $O(d_M^{1/2})$ degree and neighbor queries, and $\tilde{O}(1)$ elementary operations.

The spectrum of W is carefully tied to the spectrum of the original random walk matrix P , as was seminally proven by Szegedy in [35] and Magniez et al in [28]. For the purpose of this work, we abstract the following lemma. We say that W has a phase gap Δ if for every eigenvalue $e^{i\theta} \neq 1$ of W it holds that $|\theta| \geq \Delta$. We also recall the state $|\pi\rangle = m^{-1/2} \sum_{(i,j) \in \mathcal{E}} |i,j\rangle$.

► **Lemma 3** ([35, 28]). *Let P be the random walk transition matrix having spectral gap δ . Then the quantum walk operator W has a phase gap $\Delta \in \Theta(\sqrt{\delta})$, and $|\pi\rangle$ is the unique eigenvalue-1 eigenvector in the subspace $\text{span}_{\mathbb{C}}\{|\psi_i\rangle \mid i \in \mathcal{V}\}$.*

From this lemma, combined with the quantum algorithms for phase estimation and amplitude amplification, we can derive the folklore approach to quantum walk sampling, discussed in for instance [31, 39, 30, 29]. Since we will use it as a subroutine, we summarize it below. For a general subset $\mathcal{S} \subseteq \mathcal{V}$, we denote the state $|\mathcal{S}\rangle = d(\mathcal{S})^{-1/2} \sum_{(i,j) \in \mathcal{E}(\mathcal{S})} |i,j\rangle$.

¹ They actually consider a more general model, associated to a reversible Markov chain over G . We consider the special case where the Markov chain is a random walk.

► **Proposition 4.** *Given an initial set $\mathcal{S} \subseteq \mathcal{V}$ and a lower bound $\gamma \leq \delta$, there exists a quantum routine that generates a state ϵ -close to $|\pi\rangle$. The routine finishes and outputs a success flag after an expected number of $O(d(\mathcal{S})^{-1/2}m^{1/2}\gamma^{-1/2}\log \epsilon^{-1})$ calls to W , $O(d(\mathcal{S})^{-1/2}m^{1/2})$ reflections around $|\mathcal{S}\rangle$, and requires an additional $O(\log \epsilon^{-1} \log^2 \gamma^{-1})$ time and space complexity.*

Proof. Let the operator U be defined by the amplified quantum phase estimation algorithm, as used in [28, Theorem 6]. For some integer k , this operator maps an initial state $|\mathcal{S}\rangle$ to the state

$$U|\mathcal{S}\rangle|0\rangle = \langle \pi | \mathcal{S} \rangle |\pi\rangle |0\rangle + |\Gamma\rangle,$$

where $|\Gamma\rangle$ is such that $\|(\mathbb{I} \otimes |0\rangle\langle 0|)|\Gamma\rangle\| \leq 2^{-k}$. The operator U can be implemented using $O(k\Delta^{-1}) \in O(k\gamma^{-1/2})$ calls to W and W^\dagger , and $O(k \log^2 \gamma^{-1})$ additional space and elementary gates.

On this state we can invoke the amplitude amplification scheme from [14, Theorem 3] to retrieve the projection of $U|\mathcal{S}\rangle|0\rangle$ on the image of $\mathbb{I} \otimes |0\rangle\langle 0|$, which is 2^{-k} -close to $|\pi\rangle$. This requires an expected number of $\Theta(|\langle \mathcal{S} | \pi \rangle|^{-1})$ calls to U , U^\dagger and the reflection operator $\mathbb{I} \otimes (2|0\rangle\langle 0| - \mathbb{I})$. We prove the proposition by choosing $k \in \Theta(\log \epsilon^{-1})$ and noting that $|\langle \mathcal{S} | \pi \rangle| = d(\mathcal{S})^{1/2}m^{-1/2}$. ◀

On a general graph, and starting from some initial node $\mathcal{S} = \{i\}$, this scheme requires $\tilde{O}(d(i)^{-1/2}m^{1/2}\gamma^{-1/2}) \in \tilde{O}(m^{1/2}\gamma^{-1/2})$ QW steps, or $\tilde{O}(m^{1/2}d_M^{1/2}\gamma^{-1/2})$ degree and neighbor queries.

3 Quantum Walk Sampling

In this section we elaborate our scheme for quantum walk sampling. We separately address the process for growing a seed set, the data structure that we require, and their combination with the folklore QW sampling routine.

3.1 Growing a Seed Set

We propose the below Algorithm 1 to grow a seed set in the graph. It is a variation on the breadth-first search algorithm, returning an edge set of given size.

► **Lemma 5.** *If $M \leq m$, then Algorithm 1 outputs a subset $E \subseteq \mathcal{E}$ with $|E| \geq M$. Its time and space complexity, and degree and neighbor query complexity, are $\tilde{O}(M)$.*

Proof. Assuming the lists are ordered, any of the list and queue operations (enqueueing, dequeueing, adding an element, outputting the size of a list, searching an element in a list) takes polylogarithmic time. As a consequence, the time complexity will be determined up to log-factors by the number of for-loops before the algorithm terminates.

In every for-loop an edge is considered. Since the edges are directed, every edge is encountered at most once, at which point it is added to E . Since the algorithm terminates when $|E| = M$, this implies that the algorithm terminates after less than M for-loops. ◀

Alternatively we can output the node set $\mathcal{S} \subseteq \mathcal{V}$. Since $E \subseteq \mathcal{E}(\mathcal{S})$, we have that $d(\mathcal{S}) \geq M$.

■ **Algorithm 1** Breadth-First Edge Search.

Input: initial node i and query access to a connected graph G , integer M

Do:

- 1: create lists $S = \emptyset$ and $E = \emptyset$, and queue $B = (i)$
 - 2: **while** $B \neq \emptyset$ **do**
 - 3: $i \leftarrow \text{dequeue}(B)$, $\text{add}(S \leftarrow i)$
 - 4: **for all** j s.t. $(i, j) \in \mathcal{E}$ **do**
 - 5: **if** $j \notin S$ **then**
 - 6: $\text{add}(E \leftarrow \{(i, j), (j, i)\})$
 - 7: **if** $|E| \geq M$ **then** terminate and output E
 - 8: **if** $j \notin B$ **then** $\text{enqueue}(B \leftarrow j)$
-

3.2 Kerenidis-Prakash Data Structure

After growing the seed set $\mathcal{S} \subseteq \mathcal{V}$, we wish to use it as a resource for our QW sampling algorithm. Specifically we will require the generation of and reflection around the superposition $|\mathcal{S}\rangle$ over edges starting in \mathcal{S} . By naive query access to the database containing \mathcal{S} , this requires a time complexity $\Omega(d(\mathcal{S})^{1/2})$ per generation or reflection, which follows from the bound on index erasure [6]. Since our QW sampling algorithm will require $\Omega(m^{1/3})$ such operations, the total time complexity for $d(\mathcal{S}) \in \Theta(m^{1/3})$ would become $\Omega(m^{1/2})$, thus providing no speedup on the time complexity as compared to the folklore approach. To remedy this, we use a more efficient data structure proposed by Kerenidis and Prakash [24] in their quantum recommendation algorithm. We extract the following result, abstracted from their Theorem 15 (by setting $m = 1$, $n = n^2$ and inputting entries $(1, (i, j), 1)$ for all $(i, j) \in \mathcal{S}$).

► **Theorem 6** (Kerenidis-Prakash [24]). *Assume we have query access to a set $\mathcal{S} \subseteq \mathcal{V}$. There exists a classical data structure to store the set \mathcal{S} with the following properties:*

- *the size of the structure is $O(|\mathcal{S}| \log^2(m))$,*
- *the time and query complexity to fill the structure is $O(|\mathcal{S}| \log^2(m))$,*
- *having quantum access to the data structure we can perform the mapping $U : |0\rangle \rightarrow |\mathcal{S}\rangle$ and its inverse U^\dagger in time $\text{polylog}(m)$.*

This easily implies the ability to reflect around $|\mathcal{S}\rangle$ in time $\text{polylog}(m)$: we can rewrite the reflection $2|\mathcal{S}\rangle\langle\mathcal{S}| - \mathbb{I} = U(2|0\rangle\langle 0| - \mathbb{I})U^\dagger$, so that it comes down to implementing U , U^\dagger and a reflection around the basis state $|0\rangle$.

3.3 QW Sampling Algorithm

Building on the seed set and data structure, we can now propose our quantum sampling algorithm for creating the state $|\pi\rangle$ in $\tilde{O}(m^{1/3}\delta^{-1/3})$ time, space and quantum walk steps.

► **Theorem 7** (Quantum Walk Sampling). *If we choose $\gamma \leq \delta$ then Algorithm 2 returns a state ϵ -close to $|\pi\rangle$. The algorithm requires expected space, time and quantum walk steps in*

$$\tilde{O}(m^{1/3}\gamma^{-1/3} \log \epsilon^{-1}).$$

Proof. The correctness of the algorithm follows from Proposition 4. By this proposition we know that if $\gamma \leq \delta$ and the algorithm terminates, and hence the routine from Proposition 4 finished, then it effectively outputs a state ϵ -close to $|\pi\rangle$. The complexity of the algorithm for a fixed M is also easily bounded: the complexity of steps 2 and 3 is both $\tilde{O}(M^{1/3}\gamma^{-1/3})$,

■ **Algorithm 2** Quantum Walk Sampling.

Input: parameters γ and ϵ ; initial node i and query access to a graph G

Do:

- 1: **for** $M = 1, 2, 4, \dots, 2^k, \dots$ **do**
- 2: use BFS to grow a seed set \mathcal{S} with $d(\mathcal{S}) \in \Omega(M^{1/3}\gamma^{-1/3})$
- 3: load \mathcal{S} in data structure
- 4: apply the routine from Proposition 4 on $|\mathcal{S}\rangle$
 if the routine finishes after $\tilde{O}(M^{1/3}\gamma^{-1/3} \log \epsilon^{-1})$ steps
 then terminate algorithm and return its output
 else abort the routine and continue for-loop

which follows from Lemma 5 resp. Theorem 6. Step 4 is automatically terminated after $\tilde{O}(M^{1/3}\gamma^{-1/3} \log \epsilon^{-1})$ steps, which by Proposition 4 directly bounds the number of calls to W and reflections around $|\mathcal{S}\rangle$. By Theorem 6 the complexity of implementing a single reflection around $|\mathcal{S}\rangle$ is $\tilde{O}(1)$. The total complexity for a fixed M is therefore $\tilde{O}(M^{1/3}\gamma^{-1/3} \log \epsilon^{-1})$.

What remains to bound is the M -value at which the algorithm terminates. From Proposition 4 we know that if the number of steps $M^{1/3}\gamma^{-1/3} \log \epsilon^{-1}$ is sufficiently large, i.e.,

$$M^{1/3}\gamma^{-1/3} \log \epsilon^{-1} \in \Omega(|\langle \pi | \mathcal{S} \rangle|^{-1} \gamma^{-1/2} \log \epsilon^{-1}), \quad (2)$$

then the routine finishes with probability $\Omega(1)$. From the fact that $|\pi\rangle = m^{-1/2} \sum_{(i,j) \in \mathcal{E}} |i,j\rangle$ and $d(\mathcal{S}) \in \Omega(M^{1/3}\gamma^{-1/3})$ it holds that $|\langle \pi | \mathcal{S} \rangle| \in \Omega(M^{1/6}\gamma^{-1/6} m^{-1/2})$. As a consequence, if $M \geq m$ then $|\langle \pi | \mathcal{S} \rangle| \in \Omega(m^{-1/3}\gamma^{-1/6})$ and hence (2) will hold, such that the routine will finish with probability $\Omega(1)$. The expected number of for-loops is therefore $\log m + O(1)$, with the total complexity scaling as

$$\tilde{O}\left(\gamma^{-1/3} \log \epsilon^{-1} \sum_{k=0}^{\log m + O(1)} 2^{k/3}\right) \in \tilde{O}(m^{1/3}\gamma^{-1/3} \log \epsilon^{-1}). \quad \blacktriangleleft$$

Alternatively we are interested in bounding the algorithm in terms of classical queries. We can naively substitute every quantum walk step for $\tilde{O}(\sqrt{d_M})$ degree and neighbor queries, yielding a complexity $\tilde{O}(m^{1/3}d_M^{1/2}\gamma^{-1/3})$. However, if we are given an upper bound $D \geq d_M$, we can improve this complexity by slightly increasing the size of the seed set. We note that in the array model [20] the degrees are assumed to be known beforehand, so we exactly know d_M .

► **Corollary 8.** *Given an initial node i , a lower bound $\gamma \leq \delta$ and an upper bound $D \geq d_M$, we can generate a state ϵ -close to $|\pi\rangle$ in expected space, time, and degree and neighbor queries in*

$$\tilde{O}(m^{1/3}D^{1/3}\gamma^{-1/3} \log \epsilon^{-1}).$$

Proof. We adapt Algorithm 2 as follows: we slightly increase the size of the seed set in step 2 to $\Omega(M^{1/3}D^{1/3}\gamma^{-1/2})$, and limit the number of steps in step 4 to $\tilde{O}(M^{1/3}D^{-1/6}\gamma^{-1/3} \log \epsilon^{-1})$. Following the proof of Theorem 7, the algorithm then terminates after

$$\tilde{O}(m^{1/3}D^{1/3}\gamma^{-1/3} \log \epsilon^{-1})$$

classical steps and queries, and $\tilde{O}(m^{1/3}D^{-1/6}\gamma^{-1/3})$ QW steps. Now we can substitute each QW step with $\tilde{O}(\sqrt{d_M})$ degree and neighbor queries, yielding the claimed complexity. ◀

4 Application: st-Connectivity

4.1 General Algorithm

Let $\delta^{(s)}$ and $\delta^{(t)}$ denote the spectral gaps of the connected components of s resp. t .

► **Proposition 9.** *Given $s, t \in \mathcal{V}$ and a lower bound $\gamma \leq \delta^{(s)}, \delta^{(t)}$, we can decide st -connectivity with probability $1 - \epsilon$ in $\tilde{O}(m^{1/3}\gamma^{-1/3} \log \epsilon^{-1})$ QW steps. If we are also given an upper bound $D \geq d_M$, then we can do so in $\tilde{O}(m^{1/3}D^{1/3}\gamma^{-1/3} \log \epsilon^{-1})$ degree and neighbor queries.*

Proof. Given γ we can create an ($\epsilon' = 1/4$)-approximation $|\psi_s\rangle$ (resp. $|\psi_t\rangle$) of the superposition $|\pi^{(s)}\rangle$ (resp. $|\pi^{(t)}\rangle$) over the edges of the connected component of s (resp. t) in $\tilde{O}(m^{1/3}\gamma^{-1/3})$ QW steps. If we also have D , then we can do so in $\tilde{O}(m^{1/3}d_M^{1/3}\gamma^{-1/3})$ degree and neighbor queries.

If s and t are connected, then $|\langle \psi_s | \psi_t \rangle| \geq 1 - \epsilon'^2$, whereas if they are not, then $|\langle \psi_s | \psi_t \rangle| \leq 2\epsilon'$. We can distinguish these cases by performing the SWAP-test [2] between these states, using a single copy of both states, and $O(1)$ additional gates. If s and t are connected, then the test returns 1 with probability $(1 - |\langle \psi_s | \psi_t \rangle|)/2 \leq \epsilon'^2/2 = 1/32$, if s and t are not connected, the test returns 1 with probability $(1 + |\langle \psi_s | \psi_t \rangle|)/2 \geq 1/2 - \epsilon' = 1/4$. Repeating this scheme $O(\log \epsilon^{-1})$ times then allows to decide st -connectivity with probability $1 - \epsilon$. ◀

This approach best compares to the following classical scheme: use $\tilde{\Theta}(n^{1/2})$ independent random walks of length $\Theta(\gamma^{-1})$ from s and t to gather samples from the stationary distributions on the connected components of s resp. t . If s and t are connected then with constant probability the sample sets will overlap, which follows from the birthday paradox. This scheme requires $\tilde{O}(n^{1/2}\gamma^{-1})$ random walk steps, or equivalently, neighbor queries. It lies at the basis of the graph expansion tester by Goldreich and Ron [22], and the subsequent work on testing closeness of distributions [11] and clusterability of graphs [18].

In Table 1 we compare the query complexity of our approach to the existing quantum algorithms for st -connectivity. If no promise is given on negative instances (such as in [23] in the form of a capacitance $C_{s,t}$), then all former algorithms require $\Omega(n^{1/2})$ queries when maximized over all (s, t) -pairs of the graph. As a consequence, for the graph isomorphism problem treated in the next section, they all have a $\Omega(2^{n/2})$ complexity. Our approach however has a $\tilde{O}(2^{n/3})$ complexity.

4.2 Graph Isomorphism

We consider some given n -node graph g , described by its adjacency matrix. To this graph we can associate a new regular graph $G^{(g)} = (\mathcal{V}, \mathcal{E})$ with nodes $\mathcal{V} = \{\sigma(g) \mid \sigma \in S_n\}$, consisting of permutations of the original graph nodes, and edges $\mathcal{E} = \{(h, \sigma_{i,j}(h)) \mid h \in \mathcal{V}, i, j \in [n]\}$, corresponding to all possible transpositions of two elements. We can easily prove the following.

► **Lemma 10.** *The random walk on $G^{(g)}$ has a spectral gap $\delta \in \Omega(n^{-1} \log^{-1} n)$.*

Proof. If $|\mathcal{V}| = n!$ (i.e., $g \neq \sigma(g)$ if $\sigma \neq 1$), this graph is isomorphic to the Cayley graph derived from the symmetric group with generators given by transpositions. The mixing time of a random walk on this graph is $O(n \log n)$ by a result of Diaconis and Shashahani [19], implying a lower bound on its spectral gap $\delta \in \Omega(n^{-1} \log^{-1} n)$.

If $|\mathcal{V}| < n!$, the graph is effectively an edge contraction of the random transposition graph. Following Aldous and Fill [3, Proposition 4.44], a random walk on this graph is an *induced chain* of the random walk on the symmetric group, in particular having a spectral gap lower bounded by the spectral gap of the original walk. ◀

■ **Table 1** Query complexity of st -connectivity using different quantum algorithms in different models. The array model measures the number of degree and neighbor queries; the adjacency model measures the number of pair queries (e.g., “are i and j neighbors?”); the QW model measures the number of QW steps. The quantities $d_{s,t}$ and $R_{s,t}$ denote the length of the shortest path and the effective resistance, respectively, between s and t . The quantity $C_{s,t}$ denotes the capacitance between s and t in negative instances, i.e., if s and t are disconnected then $C_{s,t}$ quantifies “how” disconnected they are.

	query complexity	model
Dürre et al [20]	$\Theta(n)$	array
Dürre et al [20]	$\Theta(n^{3/2})$	adjacency
Belovs-Reichardt [13]	$O(m^{1/2} d_{s,t}^{1/2})$	adjacency
Belovs [12]	$O(m^{1/2} R_{s,t}^{1/2}) \in O(m^{1/2} \delta^{-1/2})$	QW
Jarret et al [23]	$O(R_{s,t}^{1/2} C_{s,t}^{1/2})$	adjacency
folklore QW sampling	$O(m^{1/2} \delta^{-1/2})$	QW
this work	$\tilde{O}(m^{1/3} \delta^{-1/3})$	QW
this work	$\tilde{O}(m^{1/3} \delta^{-1/3} d_M^{1/3})$	array

Next we show how to implement a QW step on $G^{(g)}$ in $\tilde{O}(1)$ steps. By Theorem 7 we can then create a superposition over the edges of $G^{(g)}$ (or, equivalently, its nodes) in time $\tilde{O}(m^{1/3}) = \tilde{O}(2^{n/3})$, and by Proposition 9 we can solve st -connectivity (i.e. graph isomorphism) in the same time.

► **Lemma 11.** *Implementing a quantum walk on $G^{(g)}$ takes time $\tilde{O}(1)$.*

Proof. Since we may have multi-edges, corresponding to permutations that leave the input graph invariant, we will slightly alter the QW to take place on a node+coin space (as in e.g. [1, 4]) rather than on the edge space. The relevant spectral properties from Lemma 3 however remain unchanged, as is easily seen by following for instance the proof of [25]. We define the QW node+coin space, associated to the input graph g , as $\text{span}_{\mathbb{C}}\{|\sigma(g), i, j\rangle \mid \sigma \in S_n, i, j \in [n]\}$, with S_n the symmetric group of permutations. Similarly to Section 2.2, the QW operator $W = SR_{\mathcal{E}}$ consists of a reflection $R_{\mathcal{E}}$ around a subspace $\text{span}_{\mathbb{C}}\{|\psi_{\sigma(g)}\rangle \mid \sigma \in S_n\}$, now defined as

$$|\psi_{\sigma(g)}\rangle = \frac{1}{n} \sum_{i,j \in [n]} |g, i, j\rangle,$$

and the shift operator S defined by $S|g', i, j\rangle = |\sigma_{i,j}(g'), i, j\rangle$. Each of these operators can be implemented in $\tilde{O}(1)$ steps. ◀

References

1 Dorit Aharonov, Andris Ambainis, Julia Kempe, and Umesh Vazirani. Quantum walks on graphs. In *Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC)*, pages 50–59. ACM, 2001. [arXiv:quant-ph/0012090](https://arxiv.org/abs/quant-ph/0012090)

- 2 Dorit Aharonov and Amnon Ta-Shma. Adiabatic quantum state generation and statistical zero knowledge. In *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC)*, pages 20–29. ACM, 2003. [arXiv:quant-ph/0301023](#)
- 3 David Aldous and Jim Fill. Reversible Markov chains and random walks on graphs. Unfinished monograph, 2002. URL: <https://www.stat.berkeley.edu/~aldous/RWG/book.pdf>.
- 4 Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007. [arXiv:quant-ph/0311001](#)
- 5 Andris Ambainis, Andrew M Childs, and Yi-Kai Liu. Quantum property testing for bounded-degree graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 365–376. Springer, 2011. [arXiv:1012.3174](#)
- 6 Andris Ambainis, Loïck Magnin, Martin Roetteler, and Jérémie Roland. Symmetry-assisted adversaries for quantum state generation. In *Proceedings of the 26th IEEE Conference on Computational Complexity (CCC)*, pages 167–177. IEEE, 2011. [arXiv:1012.2112](#)
- 7 Reid Andersen and Yuval Peres. Finding sparse cuts locally using evolving sets. In *Proceedings of the 41st ACM Symposium on Theory of Computing (STOC)*, pages 235–244. ACM, 2009. [arXiv:0811.3779](#)
- 8 Simon Apers and Alain Sarlette. Quantum Fast-Forwarding Markov Chains and Property Testing. *Quantum Information and Computation*, 19(3&4):181–213, 2019. [arXiv:1804.02321](#)
- 9 László Babai. Local expansion of vertex-transitive graphs and random generation in finite groups. In *Proceedings of the 23rd ACM Symposium on Theory of Computing (STOC)*, volume 91, pages 164–174. ACM, 1991. [doi:10.1145/103418.103440](#).
- 10 László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the 48th ACM Symposium on Theory of Computing (STOC)*, pages 684–697. ACM, 2016. [arXiv:1512.03547](#)
- 11 Tuğkan Batu, Lance Fortnow, Ronitt Rubinfeld, Warren D Smith, and Patrick White. Testing closeness of discrete distributions. *Journal of the ACM*, 60(1):4, 2013. [arXiv:1009.5397](#)
- 12 Aleksandrs Belovs. Quantum walks and electric networks. [arXiv:1302.3143](#), 2013.
- 13 Aleksandrs Belovs and Ben W Reichardt. Span programs and quantum algorithms for st-connectivity and claw detection. In *Proceedings of the 20th European Symposium on Algorithms (ESA)*, pages 193–204. Springer, 2012. [arXiv:1203.2603](#)
- 14 Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002. [arXiv:quant-ph/0005055](#)
- 15 Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum algorithm for the collision problem. *ACM SIGACT News (Cryptology Column)*, 28:14–19, 1997. [arXiv:quant-ph/9705002](#)
- 16 Chris Cade, Ashley Montanaro, and Aleksandrs Belovs. Time and space efficient quantum algorithms for detecting cycles and testing bipartiteness. [arXiv:1610.00581](#), 2016.
- 17 Flavio Chiericetti, Anirban Dasgupta, Ravi Kumar, Silvio Lattanzi, and Tamás Sarlós. On sampling nodes in a network. In *Proceedings of the 25th International Conference on World Wide Web (WWW)*, pages 471–481. International WWW Conferences, 2016. [doi:10.1145/2872427.2883045](#).
- 18 Artur Czumaj, Pan Peng, and Christian Sohler. Testing cluster structure of graphs. In *Proceedings of the 47th ACM Symposium on Theory of Computing (STOC)*, pages 723–732. ACM, 2015. [arXiv:1504.03294](#)
- 19 Persi Diaconis and Mehrdad Shahshahani. Generating a random permutation with random transpositions. *Probability Theory and Related Fields*, 57(2):159–179, 1981. [doi:10.1007/BF00535487](#).
- 20 Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006. [arXiv:quant-ph/0401091](#)
- 21 Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002. [doi:10.1007/s00453-001-0078-7](#).

- 22 Oded Goldreich and Dana Ron. On testing expansion in bounded-degree graphs. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 68–75. Springer, 2011. doi:10.1007/978-3-642-22670-0_9.
- 23 Michael Jarret, Stacey Jeffery, Shelby Kimmel, and Alvaro Piedrafita. Quantum algorithms for connectivity and related problems. In *Proceedings of the 26th European Symposium on Algorithms (ESA)*, pages 49:1–49:13. Springer, 2018. arXiv:1804.10591
- 24 Iordanis Kerenidis and Anupam Prakash. Quantum recommendation systems. In *Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 49:1–49:21. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. arXiv:1603.08675
- 25 Hari Krovi, Frédéric Magniez, Maris Ozols, and Jérémie Roland. Quantum walks can find a marked element on any graph. *Algorithmica*, 74(2):851–907, 2016. arXiv:1002.2419
- 26 David A Levin, Yuval Peres, and Elizabeth L Wilmer. *Markov chains and mixing times*. American Mathematical Society, 2017. doi:10.1090/mbk/058.
- 27 Andrew Lutomirski. Component mixers and a hardness result for counterfeiting quantum money. arXiv:1107.0321, 2011.
- 28 Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011. arXiv:quant-ph/0608026 doi:10.1137/090745854.
- 29 Davide Orsucci, Hans J. Briegel, and Vedran Dunjko. Faster quantum mixing for slowly evolving sequences of Markov chains. *Quantum*, 2:105, 2018. arXiv:1503.01334
- 30 David Poulin and Pawel Wocjan. Sampling from the thermal quantum Gibbs state and evaluating partition functions with a quantum computer. *Physical Review Letters*, 103(22):220502, 2009. arXiv:0905.2199
- 31 Peter C Richter. Quantum speedup of classical mixing processes. *Physical Review A*, 76(4):042306, 2007. arXiv:quant-ph/0609204
- 32 Alistair Sinclair. *Algorithms for random generation and counting: a Markov chain approach*. Springer Science & Business Media, 2012. doi:10.1007/978-1-4612-0323-0.
- 33 Rolando D Somma, Sergio Boixo, Howard Barnum, and Emanuel Knill. Quantum simulations of classical annealing processes. *Physical Review Letters*, 101(13):130504, 2008. arXiv:0804.1571
- 34 Daniel A Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM Journal on Computing*, 42(1):1–26, 2013. arXiv:0809.3232
- 35 Mario Szegedy. Quantum speed-up of Markov chain based algorithms. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 32–41. IEEE, 2004. arXiv:quant-ph/0401053
- 36 Joran Van Apeldoorn, András Gilyén, Sander Gribling, and Ronald de Wolf. Quantum SDP-Solvers: better upper and lower bounds. In *Proceedings of the 58th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 403–414. IEEE, 2017. arXiv:1705.01843
- 37 John Watrous. Succinct quantum proofs for properties of finite groups. In *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 537–546. IEEE, 2000. arXiv:cs/0009002
- 38 John Watrous. Quantum simulations of classical random walks and undirected graph connectivity. *Journal of Computer and System Sciences*, 62(2):376–391, 2001. arXiv:cs/9812012
- 39 Pawel Wocjan and Anura Abeyesinghe. Speedup via quantum sampling. *Physical Review A*, 78(4):042336, 2008. arXiv:0804.4259

PUFFINN: Parameterless and Universally Fast Finding of Nearest Neighbors

Martin Aumüller

IT University of Copenhagen, Denmark
maau@itu.dk

Tobias Christiani

IT University of Copenhagen, Denmark
tobc@itu.dk

Rasmus Pagh 

BARC, Copenhagen, Denmark
IT University of Copenhagen, Denmark
pagh@itu.dk

Michael Vesterli

IT University of Copenhagen, Denmark
miev@itu.dk

Abstract

We present PUFFINN, a parameterless LSH-based index for solving the k -nearest neighbor problem with probabilistic guarantees. By parameterless we mean that the user is only required to specify the amount of memory the index is supposed to use and the result quality that should be achieved. The index combines several heuristic ideas known in the literature. By small adaptations to the query algorithm, we make heuristics rigorous. We perform experiments on real-world and synthetic inputs to evaluate implementation choices and show that the implementation satisfies the quality guarantees while being competitive with other state-of-the-art approaches to nearest neighbor search. We describe a novel synthetic data set that is difficult to solve for almost all existing nearest neighbor search approaches, and for which PUFFINN significantly outperform previous methods.

2012 ACM Subject Classification Mathematics of computing → Probabilistic algorithms; Theory of computation → Design and analysis of algorithms; Theory of computation → Nearest neighbor algorithms

Keywords and phrases Nearest Neighbor Search, Locality-Sensitive Hashing, Adaptive Similarity Search

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.10

Related Version <https://arxiv.org/pdf/1906.12211.pdf>

Supplement Material <https://github.com/puffinn/esa-paper>

Funding The research leading to these results has received funding from the European Research Council under the European Union's 7th Framework Programme (FP7/2007-2013) / ERC grant agreement no. 614331. BARC, Basic Algorithms Research Copenhagen, is supported by the VILLUM Foundation grant 16582.

Rasmus Pagh: Supported by Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.



© Martin Aumüller, Tobias Christiani, Rasmus Pagh, and Michael Vesterli;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 10; pp. 10:1–10:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

1.1 Our results

The k -nearest neighbor (k -NN) problem has been an object of intense research, both from theoretical and applied computer scientists. There exist many implementations of k -NN data structures that perform very well in specific scenarios (see the related work section), but all implementations that we are aware of suffer from one or more of the following drawbacks:

- Not scalable to large, high-dimensional data sets.
- Not runtime-robust in the sense that query time may degrade to that of a linear search even for input distributions that are known to allow search in sublinear time.
- Not recall-robust in the sense that there are input distributions that obtain low (less than 50%) recall.
- Performance bounds only hold for well-chosen values of certain parameters that depend on the data set as well as the query distribution.

PUFFINN combines several insights from recent theoretical research on k -NN data structures into a data structure that addresses these drawbacks. Our contributions are as follows:

1. we present a parameterless and universal locality-sensitive hashing-based (LSH) implementation that solves the k -NN problem with probabilistic guarantees (Section 3)
2. we prove the correctness of an adaptive query mechanism building on top of the LSH forest data structure described by Bawa et al. in [5] (Section 3)
3. we describe an adaptive filtering approach to decrease the number of expensive distance computations (Section 4)
4. we propose a difficult dataset for the 1-NN problem that exposes weaknesses in known heuristics (Section 5)
5. we provide a detailed experimental study of our approach, evaluating design choices and relating it to the performance of other state-of-the-art approaches to k -NN search (Section 5)

Prior to this work, only subsets of these ideas have been implemented. Our main contribution on the theoretical side is that we make certain heuristics, such as the query algorithm described in [5], rigorous. While some ideas of adaptive query algorithms have been discussed before [18], they made assumptions on the data and query distribution; our methods work for k -NN search in its full generality. On the practical side, we shed light on the empirical performance of theoretical ideas with regard to possible speed-ups of an LSH implementation. Our final implementation is parameterless in the sense that it only requires the user to specify the space available for the data structure and the required quality guarantee. Our implementation is competitive to state-of-the-art k -NN algorithms; in particular, it is as performant as previous LSH solutions that are not parameterless and do not provide guarantees on the quality of the result.

1.2 Related work

There exist many fundamentally different approaches to nearest neighbor search. Popular techniques range from approximate tree-based methods, such as kd -trees [6] and M -trees [14] with an early stopping criterion [33], to random projection trees [16], to graph-based approaches [24, 21], and finally hashing-based approaches, for example using LSH [20]. Each paradigm comes with performant implementations and we will introduce some of them in

Section 5 when we are evaluating our implementation. Since the focus of the present paper is designing a provably correct LSH implementation we will focus on existing methods in this realm. See the benchmarking paper by Aumüller et al. [4] for a more detailed overview over other approaches and their performance on real-world datasets.

Locality-sensitive hashing. LSH was introduced by Indyk and Motwani in [20]. We sketch the basic idea here. An LSH data structure consists of several independent repetitions of space partitions using LSH functions. An LSH function maps a data point to a hash code such that closer points are more likely to collide than far away points. In the LSH framework, $K \geq 1$ locality-sensitive hash functions are concatenated to increase the gap between the collision probability of “close” points and “far away” points. For solving the (c, r) -near neighbor problem [20], a certain concatenation length K is fixed according to the number of points in the data set, the approximation factor c , and the strength of the hash family at hand. From the value K and the hash family one can compute how many repetitions L (using independent hash functions) have to be made to guarantee that a close point is found with constant probability. The theoretical literature on LSH has mostly focused on solving the approximate near neighbor problem which can be used to solve the approximate *nearest* neighbor problem through a reduction [19]. In this paper we use LSH to solve the exact k -nearest neighbor problem with probabilistic guarantees.

LSH implementations. Implementations of LSH evolved over the years with new advances in the theory of LSH. One of the first popular implementations, dubbed E2LSH [2], was tailored for Euclidean space and included automatic parameter tuning of the K parameter based on subsampling the dataset. From this K parameter, other parameter such as the number of repetitions are derived. It solves the problem of reporting all points within a distance r (specified during preprocessing) from the query and uses potentially large space on large datasets [2]. The multiprobing approach introduced by Dong et al. in [18] allowed for implementations in which the space parameter can be fixed as in our approach. Their parameter tuning relies on the assumption that there is a certain distance distribution between queries and data points. LSHkit was the first implementation using this idea [17]. A new LSH family for angular distance on the unit sphere motivated the development of the FALCONN library [1]. It contains highly optimized routines for efficient hash function computation, and supports multiprobing.

None of these approaches give guarantees on the query procedure if the query set is different with respect to distance distributions from the one seen during index building. Our data structure can be seen as a modified version of the LSH forest introduced by Bawa et al. in [5]. Instead of using a single hash length K , an individual repetition is a trie built on the hash codes of the data points. Our query algorithm replaces the heuristic candidate collection of a predefined size with a rigorous termination criterion.

The cost of evaluating an LSH function differs widely. It ranges from $O(1)$ time for the bitsampling approach in Hamming space [20], over $O(d)$ for random hyperplane hashing [10], to $O(d^2)$ for cross-polytope LSH [1]. For the latter, the authors of [1] proposed a heuristic version that decreases the running time to $O(d \log d)$. Another approach to reduce hashing time is to build a hashing oracle that returns the necessary KL hash function values necessary to query the LSH data structure, but builds those from a smaller set of independent hash functions. Christiani [12] described two approaches that reduce the amount of independent hash functions needed to produce these hash values from KL to $K\sqrt{L}$ (using tensoring as in [3]) or $O(\log^2 n)$ (using the pooling approach in [15]). While the E2LSH framework uses a variant of tensoring, we are not aware of an implementation using the pooling strategy.

Another idea that is currently missing in existing LSH implementations is the use of sketches, i.e., small representations of the original data points that allow to estimate the distance between two data points via their sketches. Christiani [12] describes how to use sketches when solving the near neighbor problem, but we are not aware of existing LSH-based implementations using this idea. We remark that sketching is a well-known technique and refer to the survey [27].

Auto Tuning Approaches. Apart from the approaches mentioned above, FLANN [26] and the implementation of vantage point trees in nmslib [8] are two non-LSH based nearest neighbor search that promise to tune the data structure to guarantee a certain quality criterion. This criterion is usually the recall of the query, i.e., the fraction of true nearest neighbor among the points returned by the implementation.

FLANN contains a collection of tree-based methods. For auto-tuning, it takes a small sample of the data structure and builds indexes in a certain parameter space. It then queries the data structure with points from the data set and picks, among all the indexes that achieve at least the recall the user wishes for, the one with fastest query times. The auto tuning employed by nmslib for the vantage point tree implementation follows the same principles and explores a certain parameter space based on a model of the data set to be indexed. Both approaches require that the query and data set distribution are not too different. We will see in the experiments that both of the approaches do not satisfy the recall guarantees, even on real-world datasets.

2 Preliminaries

2.1 Problem Definition

We assume a distance space (X, dist) with distance measure $\text{dist}: X \times X \rightarrow \mathbb{R}_{\geq 0}$.

► **Definition 1.** *Given a dataset $S \subseteq X$ and an integer $k \geq 1$, the (k, δ) nearest neighbor problem $((k, \delta)$ -NN) is to build a data structure, such that for every query $q \in X$, the query algorithm returns a set of k distinct points, each one being with probability at least $1 - \delta$ among the k points in S closest to q .*

An algorithm solving the (k, δ) -NN problem guarantees an expected recall of $(1 - \delta)k$, which is usually the quality measure in the context of nearest neighbor search algorithms.

2.2 Locality-Sensitive Hashing

► **Definition 2** (LSH Family [20, 10]). *A locality-sensitive hash (LSH) family \mathcal{H} is family of functions $h: X \rightarrow R$, such that for each pair $x, y \in X$ and a random $h \in \mathcal{H}$, for arbitrary $q \in X$, whenever $\text{dist}(q, x) \leq \text{dist}(q, y)$ we have $p(q, x) := \Pr[h(q) = h(x)] \geq \Pr[h(q) = h(y)]$.*

Traditionally, LSH families are used in the LSH framework to solve the (c, r) -near neighbor problem.

While the theory provided in this paper applies to every distance space that encompasses an LSH family, we set our focus on solving the k -NN problem on the d -dimensional unit sphere under angular distance, which is equivalent to cosine similarity and inner product similarity on unit length vectors. By using the Gaussian kernel approximation method of Rahimi and Recht [29] as described by Christiani in [11], our results extend to the whole Euclidean space.

Random hyperplane (HP) LSH described by Charikar in [10] and Cross-Polytope (CP) LSH introduced by Terasawa and Tanaka [31] and analyzed by Andoni et al. in [1] are two different LSH schemes under this distance measure. A single HP LSH function produces a single bit. It works by choosing a random d -dimensional vector $a = (a_1, \dots, a_d)$ where each a_i is an independent standard normal random variable. The hash code of a point x is 1 if the inner product between a and x is at least 0, and 0 otherwise. A single CP LSH function applies a random rotation of x on the unit sphere and then maps it to the index of the closest vector out of the $2d$ signed standard basis vectors. Technically, it can be thought of as choosing d random hyperplanes a_1, \dots, a_d and mapping x to the index of the hyperplane with the largest absolute inner product, separating the two cases that the inner product is negative or not.

3 Data Structure

This section describes the basic ideas of the data structure used in our implementation. Due to space reasons we only highlight the basic data structure and some of its properties. The full description with all proofs can be found in the extended version of this paper. Note that the implementation has many differences to this clean version. These differences are discussed in Section 4.

3.1 Description

In this section we will assume that we can perform distance computations and evaluate locality-sensitive hash functions in constant time. Our data structure will be parameterized by integers $L, K \geq 1$ and will consist of a collection of L LSH tries of max depth K . This data structure is known as an LSH Forest [5]. Here we use a variant of the LSH tries with bounded depth and for completeness we include a brief description and statement of relevant properties.

LSH tries. We index the LSH tries by $j = 1, \dots, L$. The j th LSH trie is built from the set of strings $\{(h_{1,j}(x), \dots, h_{K,j}(x)) \mid x \in S\}$ where $h_{i,j} \sim \mathcal{H}$. The trie is constructed by recursively splitting the set of points S on the next (i th) character until $|S| \leq i$ or $i = K + 1$ at which point we create a leaf node in the trie that stores references to the points in S . Internal nodes store pointers to its children in a hash table where the keys are locality-sensitive hash values.

Query algorithm. Let $S_{i,j}(q)$ denote the subset of points in S that collide with q when we consider the first i hash values used in the construction of the j th trie. That is, $S_{i,j}(q) = \{x \in S \mid h_{1,j}(q) = h_{1,j}(x) \wedge \dots \wedge h_{i,j}(q) = h_{i,j}(x)\}$. For our query algorithm we wish to retrieve the points in each trie that collide with our query point in a bottom-up fashion, starting at depth $i = K$. Define $\Pi_{i,j}(q) = S_{i,j}(q) \setminus S_{i+1,j}(q)$ where $S_{K+1,j}(q) = \emptyset$.

► **Fact 1.** *LSH tries have expected construction time $O(nK)$ and use $O(n)$ words of space. For $i \in \{0, 1, \dots, K\}$ we can retrieve a set $S'_{i,j}(q) \supseteq S_{i,j}(q)$ with $|S'_{i,j}(q)| \leq |S_{i,j}(q)| + i$ using time $O(|S_{i,j}(q)| + i)$. After having retrieved $S'_{i,j}(q)$ we can retrieve $S'_{i-1,j}(q)$ using additional time $O(|\Pi_{i-1,j}(q)|)$.*

The query algorithm is described in Algorithm 1. During a query we search through the LSH Forest starting with the buckets $S_{i,j}(q)$ at depth $i = K$ and moving up one level once the L LSH tries have been explored at the current level. While searching we use a data structure PQ to keep track of the top- k closest points seen so far. We stop the search once

■ **Algorithm 1** ADAPTIVE-KNN(q, k, δ).

```

1 PQ ← empty priority queue of (point, dist) of unique points
2 for  $i \leftarrow K, K-1, \dots, 0$  do
3   for  $j \leftarrow 1, 2, \dots, L$  do
4     for  $x \in \Pi_{i,j}(q)$  do
5       /* We abbreviate  $x'_k \leftarrow \text{PQ.max}()$  for ease of notation */
6       if  $\text{dist}(q, x'_k) \geq \text{dist}(q, x)$  then
7         PQ.insert( $x, \text{dist}(q, x)$ ) // Remove largest entry if PQ contains
8         more than  $k$  elements.
9       end
10      end
11      if  $i = 0$  or  $(\text{PQ.size}() == k \text{ and } j \geq \ln(1/\delta)/p(q, x'_k)^i)$  then
12        return PQ
13      end
14    end
15  end
16 end

```

we have searched sufficiently many tries at a depth where our current k -nearest neighbor candidate would have been found with probability at least $1 - \delta$. This stopping criterion ensures that we always search far enough to find the true k -nearest neighbor with probability at least $1 - \delta$.

Insertions (Line 6) and retrieval of the k -largest distance (Lines 5 and 9) can be done in expected amortized time $O(1)$ by using an array of size $2k$ which is updated after each k insertions. For simplicity the algorithm is described as a double for-loop that iterates over the sets $\Pi_{i,j}(q)$ in a bottom-up fashion. Using LSH tries Algorithm 1 would be implemented by using a straight-forward bottom-up traversal of the tries with properties described in Fact 1.

We proceed by proving that Algorithm 1 solves the (k, δ) -NN problem as well as providing running time bounds. The idea behind an adaptive k NN algorithm with guarantees following the approach of Algorithm 1 can be attributed to Dong et al. [18]. Christiani et al. [13] show how a different stopping criteria gives a self-tuning algorithm in the regime where $\delta = 1/n$. To the best of our knowledge both the following proof of correctness (although simple) and the running time bound for Algorithm 1 is new.

► **Lemma 3.** ADAPTIVE-KNN(q, k, δ) returns a set of k points, each one being with probability at least $1 - \delta$ among the closest k points to q .

Proof. As introduced in Definition 2, we use the short notation $p(q, x) := \Pr[h(q) = h(x)]$ under the random LSH hash function choice h . Let x_1, \dots, x_k be the k closest neighbors of q in S . First, observe that at any stage the algorithm maintains the invariant “PQ.size() < k or $p(q, x'_k) \leq p(q, x_k)$ ”. This is true because at any point, the k -th closest point x'_k identified by the algorithm satisfies $\text{dist}(q, x'_k) \geq \text{dist}(q, x_k)$. Together with the monotonicity of the collision probability of LSH, cf. Definition 2, the invariant holds. Thus, the algorithm cannot terminate until j' tries have been searched at level i' where either $j' \geq \ln(1/\delta)/p(q, x_k)^{i'}$ or $i' = 0$. In the first case the probability of not finding a k NN of q is at most $(1 - p(q, x_k)^{i'})^{j'} \leq \delta$. In the case of $i' = 0$ the query algorithm degrades to a linear scan and we are guaranteed to report the true k NNs. ◀

Next, we connect the expected running time of Algorithm 1 to the optimal expected running time of an algorithm that knows optimal parameter choices for i and j . We will use $OPT(L, K, k, \delta)$ to denote the optimal expected query time that can be achieved with the natural algorithm that solves k NN queries on the LSH Forest by searching j tries at depth i where i and j are chosen to minimize the query time. In our expression for the expected the query time we use a unit cost model that counts hash function evaluations and distance computations. To ensure that each point in the k NN set is reported with probability at least $1 - \delta$ we search $j = \ln(1/\delta)/p(q, x_k)^i$ tries. The expected cost of searching one LSH trie at depth i is $i + \sum_{x \in P} p(q, x)^i$.

$$OPT(L, K, k, \delta) = \min \left\{ \frac{\ln(1/\delta)}{p(q, x_k)^i} (i + \sum_{x \in P} p(q, x)^i) \mid 0 \leq i \leq K, \frac{\ln(1/\delta)}{p(q, x_k)^i} \geq L \right\}.$$

We obtain the following lemma with a proof provided in the extended version of this paper.

► **Lemma 4.** *Let $0 < \delta \leq 1/2$ then with probability $1 - \delta$ we have that $\text{ADAPTIVE-KNN}(q, k, \delta)$ terminates in expected time $O(OPT(L, K, k, \delta/k) + L(K + k))$.*

3.2 Reducing the Number of LSH Evaluations

In [12], Christiani provides a uniform framework encompassing previous ad-hoc solutions [15, 3] to reduce the amount of hash function evaluations when solving the approximate near neighbor problem. In the extended version of this paper we describe how these techniques can be applied when solving the k -NN problem. In particular, each method requires an adaption of Algorithm 1 and comes with their own stopping criterion. We provide a succinct description of the methods next.

Tensoring. Assume that K is an even integer and L is an even power of two. Form two collections of \sqrt{L} tuples of $K/2$ LSH functions. Each trie in the LSH forest is now indexed by $j_1, j_2 \in \{1, \dots, \sqrt{L}\}$. The K LSH functions used in the (j_1, j_2) st trie are taken by interleaving the $K/2$ functions in the j_1 st tuple of the first collection and the j_2 nd tuple of the second collection. This allows us to construct L LSH tries of max depth K using only $\sqrt{L}K$ independent functions.

Pooling. Form a pool of m independent LSH functions that will be shared among LSH tries. For each LSH trie in the LSH Forest we independently sample a random subset of K LSH functions from the pool that we use in place of fully random LSH functions. The LSH pool can be viewed as a randomized construction of a smaller LSH family from our original LSH family. As m increases LSH functions sampled without replacement from the pool will work almost as well as independent samples from the LSH family.

3.3 Sketching for faster distance computations

Locality-sensitive hashing can be used to produce 1-bit sketches for efficient similarity estimation [10, 23]. The idea is to use a random hash function to hash the output of a locality-sensitive hash function to a single bit, and then packing w such bits into a w -bit machine word. We can use word parallelism (alternatively table lookups) to count the number of collisions between $w = \Theta(\log n)$ such sketches in $O(1)$ time, allowing us to efficiently estimate the similarity between points in our original space. Depending on the LSH scheme used and the distribution of distances between the query point and the data, using 1-bit sketches can replace many of the expensive distance computations performed by the query algorithm with cheaper distance estimations through sketching. See [30, 12] for more details on sketching in the context of the ANN problem.

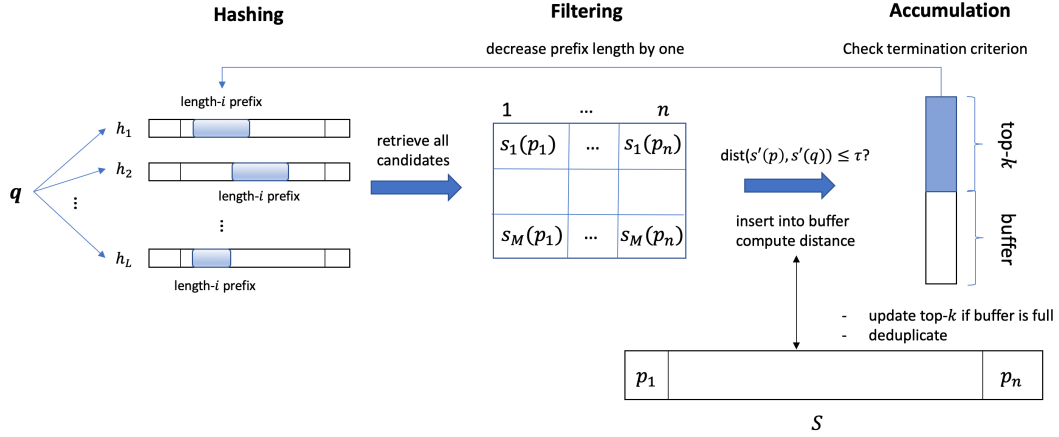


Figure 1 Overall structure of our implementation.

4 Implementation Overview

4.1 Overall structure

Figure 1 presents an overview over our data structure. Deviating from the pointer-based trie data structure described in Section 3, we use an array of indices sorted by hash code, which improves both cache- and space-efficiency. Additionally, a sketching-based filtering approach is used to reduce the number of distance computations carried out during a query. In the following, we make the implementation precise.

A query is answered in rounds, starting at the maximum considered prefix length. A single round works as follows: Repetitions are inspected one after the other. In each repetition, all (new) candidates sharing the hash prefix with the query point are retrieved. For each such candidate point, a sketch is chosen and checked against the corresponding sketch of the query point. Let τ denote a threshold value that we will discuss how to set later. If the Hamming distance between candidate and query sketch is less than τ , the data point passes the filter, the distance computation is carried out, and the point with its distance to the query is inserted into the accumulator buffer. Once this buffer is full we discard all points not belonging to the top- k . While this can be done in time $O(k)$, we found that an implementation based on sorting was faster for the values of k we considered. The termination criterion is checked after all repetitions are inspected. If the criterion is satisfied, the algorithm returns the indices of the top- k points in the accumulator, otherwise the prefix length is decreased by one and a new round starts.

The following subsections make this process more detailed and discuss engineering choices in the case of angular distance on unit length vectors.

4.2 Engineering choices

Vector storage. We normalize the query vector and all data vectors, which means that all dot products will be between -1 and 1. This allows storing vectors in a fixed point format represented using 16-bit integers. Such a representation enables AVX2-enabled instructions that allow 16 multiplications at once. To use the AVX instructions, all vectors are stored in a 1-dimensional array with padding to ensure 256-bit alignment.

Retrieving candidate points. The j th LSH repetition is represented as a sorted array of tuples of the form $(h_{\leq i,j}(p), \text{pos}(p))$, where $\text{pos}(p)$ is the index of $p \in S$ in the original dataset, and $h_{\leq i,j}(p)$ is the hash code of p under hash functions $h_{1,j} \circ \dots \circ h_{i,j}$. We view the hash code as a bitstring. Before retrieving candidates, we first find the tuple with the longest common prefix with the hash of the query vector. This is achieved using binary search, where we tabulate the lexicographic position of each 13-bit prefix to speed up the search.

Each time more candidates are requested, all tuples whose hash code has a common prefix of length at least i are considered. Each iteration decrements i in order to increase the number of considered vectors. Since the vectors are stored in sorted order, all tuples with a common prefix of length i are stored adjacent to the tuples with common prefixes of length $i + 1$. Furthermore, they are all stored on the same side, depending on whether the removed bit was a 0 or 1. This means that the range of considered vectors can be updated efficiently.

Every access in the array is done in a segment of size $B = 12$, regardless of whether the prefix matches or not. This costs almost no time, because the random memory access is the expensive part, and only improves quality. A discussion of suitable values of B is provided in the extended version of this paper.

Filtering candidate points. The filtering step is an additional measure to reduce the number of distance computations. Fix a point $p \in S$. During the index building phase, we store M 64-bit sketches $s_1(p), \dots, s_M(p)$ obtained via HP LSH. If p is retrieved as a candidate, retrieve a sketch $s'(p)$ using a pseudorandom transformation of the repetition number j . Next, compute the Hamming distance between $s'(p)$ and $s'(q)$. If the distance is at most $\tau \in \{0, \dots, b\}$, p passes the filter and is inserted into the accumulator. A challenge in the context of k -NN queries is that the algorithm does not know the distance to the k -th nearest neighbor. This means that the threshold has to be adapted according to the points inspected so far. We set threshold τ dynamically according to the probability that a vector with Hamming distance τ or less has a dot product larger than the smallest dot product in the current top- k .

Computing distances. The accumulator takes care of the candidate points that pass the filter step. It de-duplicates the candidate list and keeps track of the top- k points found so far. The accumulator consists of a buffer of size $2k$, which contains the current top- k indices, along with their dot products, and a buffer of size k , which contains points that passed the filter along with their dot products. Once this buffer is full, the top- k list is updated.

4.3 Locality-sensitive Hash Functions

Supported LSH Functions. The supported hash functions relevant for the paper are HP LSH [10] and CP LSH [1].¹ For the latter, the implementation encompasses both the exact version and the pseudorandom version with three applications of the fast Hadamard transform, see [1] for more details. We always regard hash functions as producing an ℓ -bit string as its output. For HP LSH, we have $\ell = 1$, for CP LSH, we have $\ell = \lceil \log 2d \rceil$. In case the algorithm did not terminate after exploring all L repetitions, decreasing the prefix length by one always means that we disregard the last bit of the hash. This is to avoid a sudden increase in the number of collisions in the case of CP LSH. This is theoretically sound since the termination criteria from Section 3 only need a lower bound on the collision probability at a certain prefix-length, which can be estimated for individually bit lengths of CP LSH.

¹ PUFFINN is generic to the LSH family, but some engineering choices are different for other similarity measures, such as set similarity. At the moment, PUFFINN also support (b -bit) MinHash [9, 23].

Estimating Collision Probabilities. Recall from Section 3 that evaluating the collision probability of two points at a certain distance is a key ingredient in the query algorithm. While such a formula is easy to derive for HP LSH, we only know of the asymptotic behavior of collision probabilities for CP LSH [1]. To overcome this obstacle, we find a Monte Carlo estimate on the collision probability of unit vectors with inner product α , $-1 \leq \alpha \leq 1$, by enumerating different values of α in a window of size .05. For a fixed distance, we consider two points $x = (1, 0, \dots, 0)$ and $y = (\alpha, \sqrt{1 - \alpha^2}, 0, \dots, 0)$,² draw 1 000 random CP hash functions, count the number of collisions, and tabulate the estimate. As mentioned above, we always consider bit strings, so the probability estimation for CP LSH is made for all bit lengths up to $\ell = \lceil \log 2d \rceil$.

In the query procedure, we round the distance down to the closest distance value for which we have tabulated an estimate and use that to bound the collision probability. The evaluation in the next section will show that this yields a negligible loss in quality compared to an exact variant using HP LSH.

5 Experimental Evaluation

Implementation and Experimental Setup. PUFFINN is implemented in C++ and comes with a wrapper to the Python language. Experiments were run on 2x 14-core Intel Xeon E5-2690v4 (2.60GHz) with 512GB of RAM using Ubuntu 16.10 with kernel 4.4.0. It is compiled using g++ with the compiler flags `-std=c++14 -Wall -Wextra -Wno-noexcept-type -march=native -O3 -g -fopenmp`. Index building was multi-threaded, queries were answered sequentially in a single thread. The experiments were conducted in the `ann-benchmarks` framework from [4]. The code, raw experimental results, and the Jupyter notebook used for the evaluation are available at <https://github.com/puffinn/esa-paper>.

Quality and Performance Metrics. As quality metric we measure the individual recall of each query, i.e., the fraction of points reported by the implementation that are among the true k -NN. As performance metric, we record individual query times. We usually report on the *throughput*, i.e., the average number of queries that can be answered in one second. In plots, the throughput is dubbed QPS for *queries per second*.

Parameter Choices. PUFFINN has two parameters: the space a user is willing to allocate for the index, and the expected recall that should be achieved. We run PUFFINN with expected recall values in the set $\{.1, .2, .5, .7, .9, .95\}$. As space parameters, we use the doubling range 512 MB to 32 GB. We always retrieve the ten nearest neighbors.

Objectives of the Experiments. Our experiments are tailored to answer the following high-level questions (*HL-Q*):

(HL-Q1) Given the choices of sketching and hash function evaluation methods described in the previous two sections, how do they compare to each other w.r.t. empirical performance (Sections 5.1–5.3)?

(HL-Q2) Can a parameterless method compete with implementations that have parameters tuned to the data and query workload (Section 5.5)?

² By spherical symmetry, the collision probabilities are the same for all pairs of points with inner product α .

■ **Table 1** Datasets under consideration.

Dataset	Data Points/Query Points	Dimensions
GLOVE [28]	1 183 514/10 000	100
GLOVE-2M [28]	2 196 018/10 000	300
GNEWS-3M [25]	3 000 000/10 000	300
SYNTHETIC	1 000 000/ 1 000	300

To answer these questions, we will consider the following implementation-level questions (*IL-Q*):

(IL-Q1) What is the influence of the filtering approach to the quality/QPS (Section 5.1)?

(IL-Q2) What is the influence of the update threshold τ to quality and QPS (Section 5.1)?

(IL-Q3) How does the space parameter influence the QPS (Section 5.2)?

(IL-Q4) How does the hash function and evaluation strategy influence performance (Section 5.3)?

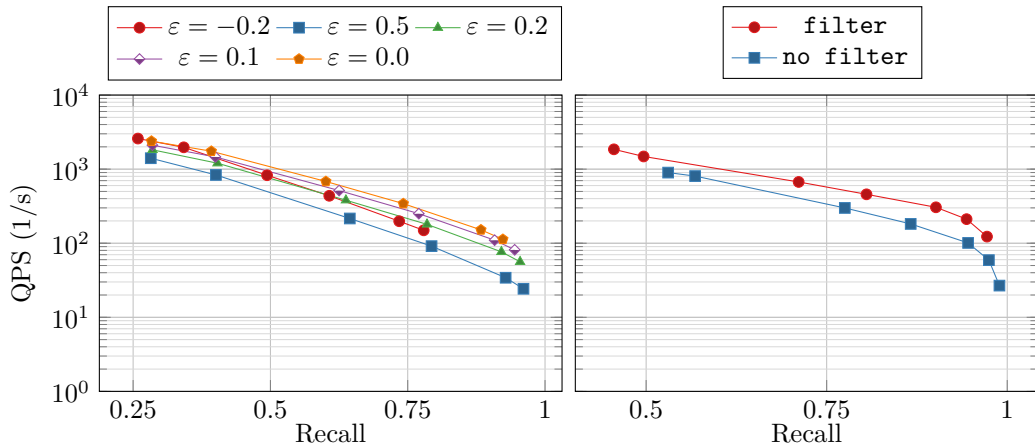
Real-World Datasets. Table 1 gives an overview of the datasets used in the experiments. Cosine similarity is usually used in the context of word embeddings, so we use three real-world datasets that originate from two different word embedding algorithms. Unless stated otherwise, all experiments are carried out on `Glove-1M`.

Synthetic Dataset. We describe a synthetic data set and query distribution that, as we will see, is challenging for many heuristic nearest neighbor implementations. For a fixed $d \geq 1$, we construct a dataset over \mathbb{R}^{3d} as follows. For each $i \in \{1, \dots, n-1\}$, let y_i and z_i be two d -dimensional vectors of expected length $\sqrt{1/2}$ where each coordinate is sampled independently from $\mathcal{N}(0, 1/2d)$. Define $x_i = (0^d, y_i, z_i)$. Let v and w be two more random vectors of expected length $\sqrt{1/2}$. Finally, set $x_n = (v, w, 0^d)$. We define m query vectors as follows: For each $i \in \{1, \dots, m\}$, let $q_i = (v, 0^d, r_i)$, where r_i is a random vector of length $\sqrt{1/2}$.

This construction has the property that x_n is the nearest neighbor of every query. Furthermore, all data points have unit length in expectation. The distance from each q_i to x_n is expected to be 1 (or equivalently $\mathbb{E}[\langle q_i, x_n \rangle] = \frac{1}{2}$), whereas the distance to all other points is around $\sqrt{2}$ (i.e., $\mathbb{E}[\langle q_i, x_j \rangle] = 0$). In the experiment, we choose $n = 1\,000\,000$ and $d = 100$.

Other approaches. We compare PUFFINN to the following implementations: FALCONN, a state-of-the-art LSH implementation using the theory developed in [1]; ONNG, a recent graph-based approach described in [21]; ANNOY, the best-performing implementation of a random-projection forest [7]; IVF, a k -means clustering based approach [22]; FLANN, a collection of different approaches with tuning of recall value [26]; VantagePointTree [32] as implemented in NMSlib [8] with recall guarantees.

These approaches stood out in the benchmarking paper from Aumüller et al. [4] as performing best on many datasets. We use the same parameter space as in [4] to test the performance of the different implementations. For each implementation, we report the best results achieved via a grid search over the (usually large) parameter space. We refer to that paper or the original papers for more details on these approaches. Except from FLANN and VantagePointTree, no other implementation allows to specify a guarantee on recall.



■ **Figure 2** Left: Influence of setting the threshold of sketches to $1 + \varepsilon$ times the expected difference at the distance of the k -th closest point found so far. Expected recall values: 0.1, 0.2, 0.5, 0.7, 0.9, 0.95, space: 1GB. Right: Difference between filtering/no filtering, space: 4GB.

5.1 Filtering Approach

We evaluate the filtering approach in two directions. First, we report on the quality-performance trade-off of different update strategies. Second, we benchmark the architecture against a “no filtering” approach. Experiments were done using a collection of 32 sketches using 64 bits each.

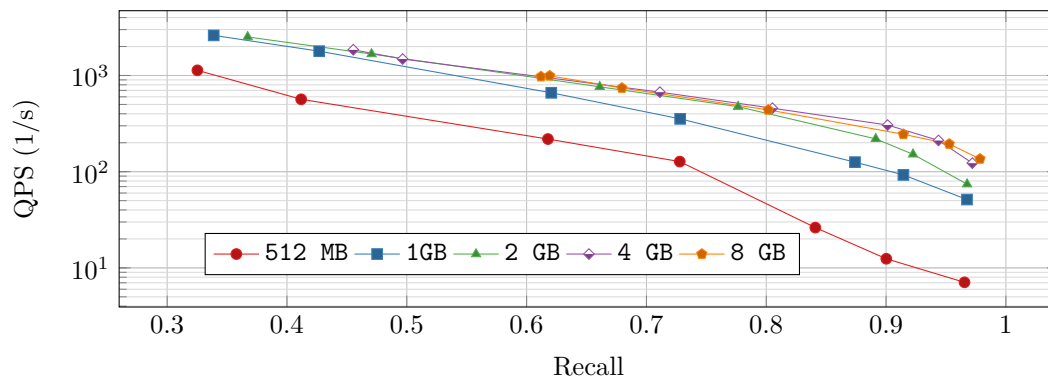
Figure 2 (top) reports on the influence of setting the passing threshold of the filtering step dynamically to a fraction of $\tau = 1 + \varepsilon$ of the expected difference³ for a point at the distance of the current k -th nearest neighbor. A ε -value of 0.0, 0.1, and 0.2 give good results in this empirical setting. Above 0.1 there is almost no gain in quality but a huge drop in QPS. Setting the threshold below the expectation results in a large loss in quality. For the remainder of the experiments, we set the threshold to the expectation, i.e., we use $\delta = 0$.

Figure 2 (bottom) allows us to see the difference between using resp. not using the filtering approach. For low recall values, the filtering approach increases the QPS by a factor of roughly 1.5. For example, the filtering approach can answer around 1 400 QPS at recall .5, without filtering this number drops to 900. At high recall, the difference is more pronounced. A recall of 97% is achieved with 122 QPS using filtering and the same recall is achieved at around 50 QPS without filtering. We can see a clear difference in the achieved quality between the two variants. The sketching approach usually decreases the recall for a fixed expected recall by .08 (for low recall) to .03 (for high recall). However, the recall is still above the guarantee in both cases.

5.2 Influence of the Index Size

We turn our focus to the third implementation-level question: How does the index size influence the performance? We note that the index size includes the whole data structure, including storage of the original dataset and the hash functions. Figure 3 reports on the quality-performance trade-off achieved by the implementation for different index sizes. We

³ The expected difference is just 64 times the probability that two points at distance r_k collide under a random hyperplane.



■ **Figure 3** Influence of index size to quality-performance trade-off.

observe that larger index sizes provide better performance, but the influence is diminishing at more than 4 GB. A small index yields a small number of repetitions, which means that the algorithm has to explore many levels in the data structure. For a recall of .9, increasing the amount of space from 512 MB to 2 GB increases the QPS from 12 to roughly 200. Doubling the allotted space to 4GB results in around 300 QPS, which is roughly the same for 8 GB as well. We can see that the achieved recall is above the set guarantee threshold for all tested index sizes. (Each data point corresponds to a recall value from the set of tested recall values.)

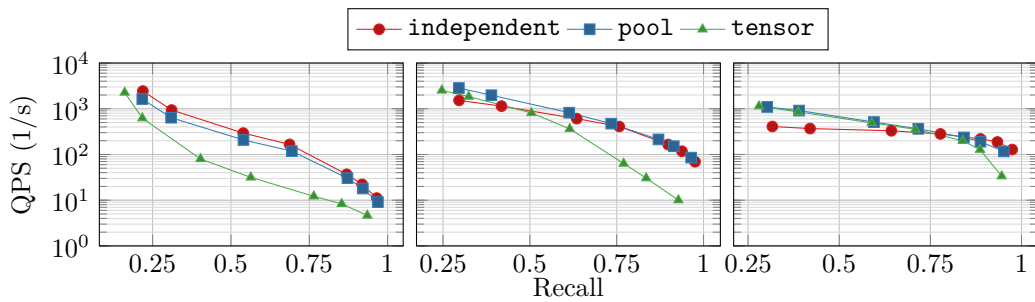
5.3 Choice of Hash Function and Evaluation Strategy

We start by evaluating different hash evaluation strategies. We implemented the following three different evaluation strategies in PUFFINN: independent, tensor, and pool. For the pooling strategy, we set up a pool containing 3072 bits. In the following, we fix the hash function used to be CP LSH using fast Hadamard transform. Figure 4 shows a comparison between the three evaluation strategies for different index sizes. As we can see, tensoring is never better than the pooling strategy. Furthermore, independent gives better performance for a fixed quality for large index sizes, but takes more time for initializing the hash values at low recall. We note that when using the exact CP LSH, there is a huge difference between independent and pooling, in particular for large index sizes. For example, CP using independent hash functions achieves not more than 80 QPS for the 8 GB index in the right plot in Figure 4. For all of these reasons, we fix the implementation to use the pooling strategy.

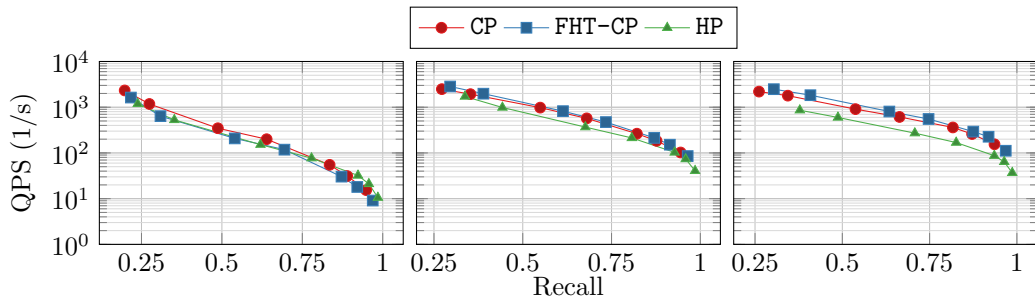
We turn our focus to the choice of hash function. Figure 5 gives an overview of three different index sizes using the pooling strategy for hash function evaluation. On the smallest index, HP LSH works well, in particular for high recall. If there is space for more repetitions, CP LSH becomes the method of choice, and FHT-CP is a bit faster than the exact method.

5.4 Summary of Implementation Choices

In light of our first high-level question, we observed that sketching provides a good performance increase at negligible cost. The larger the index size, the faster PUFFINN can answer queries. However, it works well on small index sizes as well. We fixed the pooling strategy as the evaluation strategy since it was much faster than tensoring and allows to use exact methods such as CP LSH when FHT-CP does not satisfy the guarantees one wishes for.



■ **Figure 4** Comparison of hash evaluation strategies. Left: 512 MB, center: 2 GB, right: 8 GB.



■ **Figure 5** Hash function comparison (pooling). Left: 512 MB, center: 1 GB, right: 4 GB.

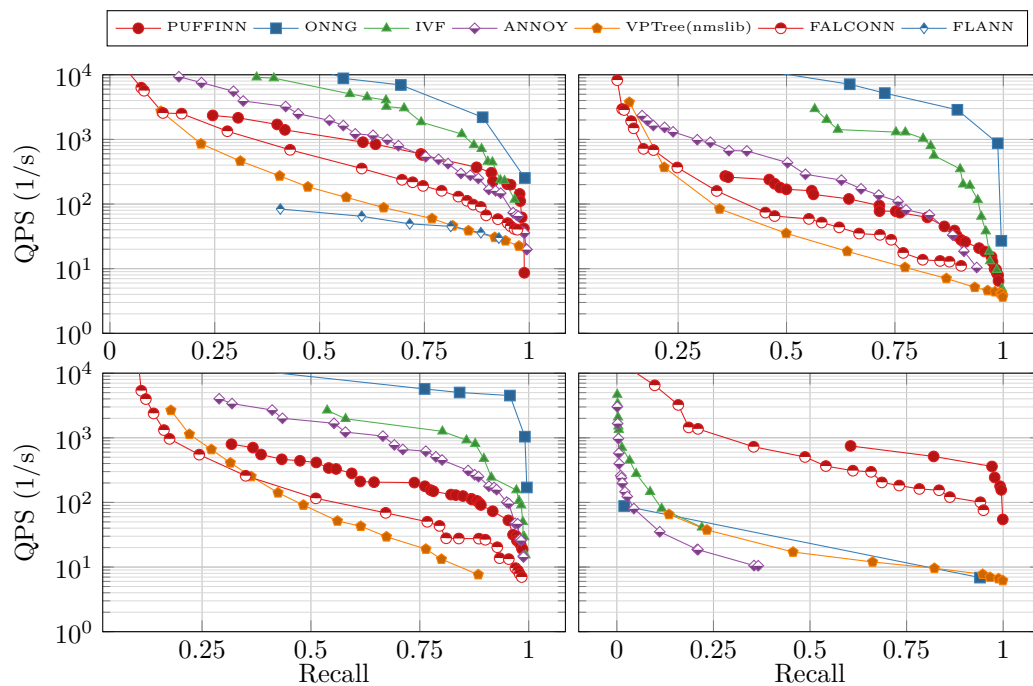
5.5 Comparison to Other Approaches

We turn our focus on comparing PUFFINN to other approaches on real-world and synthetic datasets (HL-Q2). Figure 6 gives an overview of the performance-quality trade-off achieved by state-of-the-art k -NN approaches on three real-world dataset and a synthetic one. Among the implementations that support automatic parameter tuning, PUFFINN is usually by a large factor the fastest implementation. We remark that the automatic parameter tuning of FLANN failed to build an index within 6 hours on three out of the four datasets and was disregarded on those.

PUFFINN managed to obtain at least recall .95 on every dataset, whereas the tuning of the VPtree failed to achieve high recall on GNEWS-3M. PUFFINN shows better performance than FALCONN for most of the performance-quality space. It is comparable in performance to ANNOY on most of the real-world datasets, and is particular competitive in the high-recall setting. On real-world datasets, IVF and in particular ONNG show better performance than PUFFINN except for very high recall. This is an indicator that graph-based approaches perform best on these real-world datasets (for good manual parameter choices).

On the synthetic dataset, only LSH-based approaches achieve high recall at high QPS. VPtree and ONNG manage recall close to 1, but are more than a factor 10 slower than PUFFINN. IVF and ANNOY fail to achieve recall higher than 40% on the synthetic dataset.

To come back to our second high-level question: It is possible to compete with state-of-the-art implementations of k -NN using a parameterless method with guarantees. PUFFINN is easy to use and achieves performance comparable to many of its competitors. Among LSH variants, it is as fast as FALCONN which does not give guarantees. This means that our engineering choices allowed an LSH implementation with theoretical guarantees that come “for free”.



■ **Figure 6** Comparison of different implementations with PUFFINN (index size at most 8 GB). Top left: Glove-1M, top right: Glove-2M, bottom left: GNEWS-3M, bottom right: Synthetic.

References

- 1 A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt. Practical and optimal LSH for angular distance. In *Proc. NIPS '15*, pages 1225–1233, 2015.
- 2 Alexandr Andoni and Piotr Indyk. E2LSH, user manual, 2005.
- 3 Alexandr Andoni and Piotr Indyk. Efficient algorithms for substring near neighbor problem. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1203–1212. Society for Industrial and Applied Mathematics, 2006.
- 4 Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms. In *SISAP'17*, pages 34–49, 2017. doi:10.1007/978-3-319-68474-1_3.
- 5 Mayank Bawa, Tyson Condie, and Prasanna Ganesan. LSH forest: self-tuning indexes for similarity search. In *Proceedings of 14th International Conference on World Wide Web (WWW)*, pages 651–660. ACM, 2005.
- 6 Jon Louis Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM*, 18(9):509–517, 1975.
- 7 Erik Bernhardsson. Annoy. URL: <https://github.com/spotify/annoy>.
- 8 Leonid Boytsov and Bilegsaikhan Naidan. Engineering Efficient and Effective Non-metric Space Library. In *SISAP'13*, pages 280–293, 2013. doi:10.1007/978-3-642-41062-8_28.
- 9 Andrei Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.
- 10 Moses Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of 34th ACM Symposium on Theory of Computing (STOC)*, pages 380–388, 2002. doi:10.1145/509907.509965.
- 11 Tobias Christiani. A Framework for Similarity Search with Space-Time Tradeoffs using Locality-Sensitive Filtering. In *Proc. 28th Symposium on Discrete Algorithms (SODA)*, pages 31–46, 2017.

10:16 PUFFINN: Parameterless and Universally Fast Finding of Nearest Neighbors

- 12 Tobias Christiani. Fast Locality-Sensitive Hashing for Approximate Near Neighbor Search. *CoRR*, abs/1708.07586, 2017. [arXiv:1708.07586](https://arxiv.org/abs/1708.07586).
- 13 Tobias Christiani, Rasmus Pagh, and Mikkel Thorup. Confirmation Sampling for Exact Nearest Neighbor Search. *CoRR*, abs/1812.02603, 2018. [arXiv:1812.02603](https://arxiv.org/abs/1812.02603).
- 14 Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB'97*, pages 426–435, 1997. URL: <http://www.vldb.org/conf/1997/P426.PDF>.
- 15 Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Mikkel Thorup. Fast Similarity Sketching. *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 663–671, 2017.
- 16 Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *STOC'08*, pages 537–546. ACM, 2008.
- 17 Wei Dong. LSHkit. URL: <http://lshkit.sourceforge.net/>.
- 18 Wei Dong, Zhe Wang, William Josephson, Moses Charikar, and Kai Li. Modeling LSH for performance tuning. In *Proceedings of 17th ACM Conference on Information and Knowledge Management (CIKM)*, pages 669–678. ACM, 2008.
- 19 Sarel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. *Theory of Computing*, 8(1):321–350, 2012. doi:10.4086/toc.2012.v008a014.
- 20 Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of 30th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 604–613, 1998. doi:10.1145/276698.276876.
- 21 M. Iwasaki and D. Miyazaki. Optimization of Indexing Based on k-Nearest Neighbor Graph for Proximity Search in High-dimensional Data. *ArXiv e-prints*, October 2018. [arXiv:1810.07355](https://arxiv.org/abs/1810.07355).
- 22 Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *CoRR*, abs/1702.08734, 2017. [arXiv:1702.08734](https://arxiv.org/abs/1702.08734).
- 23 Ping Li and Arnd Christian König. Theory and applications of b -bit minwise hashing. *Commun. ACM*, 54(8):101–109, 2011. doi:10.1145/1978542.1978566.
- 24 Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *ArXiv e-prints*, March 2016. [arXiv:1603.09320](https://arxiv.org/abs/1603.09320).
- 25 Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *CoRR*, abs/1301.3781, 2013. [arXiv:1301.3781](https://arxiv.org/abs/1301.3781).
- 26 Marius Muja and David G. Lowe. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. In *VISSAPP'09*, pages 331–340. INSTICC Press, 2009.
- 27 Rasmus Pagh. Similarity Sketching. In *Encyclopedia of Big Data Technologies*. Springer, 2019. doi:10.1007/978-3-319-63962-8_58-1.
- 28 Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.
- 29 Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008.
- 30 Venu Satuluri and Srinivasan Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. *Proceedings of the VLDB Endowment*, 5(5):430–441, 2012.
- 31 Kengo Terasawa and Yuzuru Tanaka. Spherical LSH for Approximate Nearest Neighbor Search on Unit Hypersphere. In *Algorithms and Data Structures, 10th International Workshop, WADS 2007, Halifax, Canada, August 15-17, 2007, Proceedings*, pages 27–38, 2007. doi:10.1007/978-3-540-73951-7_4.
- 32 Peter N. Yianilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA.*, pages 311–321, 1993. URL: <http://dl.acm.org/citation.cfm?id=313559.313789>.
- 33 Pavel Zezula, Pasquale Savino, Giuseppe Amato, and Fausto Rabitti. Approximate Similarity Retrieval with M-Trees. *VLDB J.*, 7(4):275–293, 1998. doi:10.1007/s007780050069.

Online Multistage Subset Maximization Problems

Evrpidis Bampis

Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
evripidis.bampis@lip6.fr

Bruno Escoffier

Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
bruno.escoffier@lip6.fr

Kevin Schewior

Fakultät für Informatik, Technische Universität München, Germany
Département d'Informatique, École Normale Supérieure Paris, PSL University, France
kschewior@gmail.com

Alexandre Teiller

Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
alexandre.teiller@lip6.fr

Abstract

Numerous combinatorial optimization problems (knapsack, maximum-weight matching, etc.) can be expressed as *subset maximization problems*: One is given a ground set $N = \{1, \dots, n\}$, a collection $\mathcal{F} \subseteq 2^N$ of subsets thereof such that $\emptyset \in \mathcal{F}$, and an objective (profit) function $p : \mathcal{F} \rightarrow \mathbb{R}_+$. The task is to choose a set $S \in \mathcal{F}$ that maximizes $p(S)$. We consider the *multistage* version (Eisenstat et al., Gupta et al., both ICALP 2014) of such problems: The profit function p_t (and possibly the set of feasible solutions \mathcal{F}_t) may change over time. Since in many applications changing the solution is costly, the task becomes to find a sequence of solutions that optimizes the trade-off between good per-time solutions and stable solutions taking into account an additional similarity bonus. As similarity measure for two consecutive solutions, we consider either the size of the intersection of the two solutions or the difference of n and the Hamming distance between the two characteristic vectors.

We study multistage subset maximization problems in the *online* setting, that is, p_t (along with possibly \mathcal{F}_t) only arrive one by one and, upon such an arrival, the online algorithm has to output the corresponding solution without knowledge of the future.

We develop general techniques for online multistage subset maximization and thereby characterize those models (given by the type of data evolution and the type of similarity measure) that admit a constant-competitive online algorithm. When no constant competitive ratio is possible, we employ lookahead to circumvent this issue. When a constant competitive ratio is possible, we provide almost matching lower and upper bounds on the best achievable one.

2012 ACM Subject Classification Theory of computation \rightarrow Approximation algorithms analysis; Theory of computation \rightarrow Online algorithms

Keywords and phrases Multistage optimization, Online algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.11

Related Version A full version of the paper is available at [7], <http://arxiv.org/abs/1905.04162>.

Funding *Kevin Schewior*: Supported by the DAAD within the PRIME program using funds of BMBF and the EU Marie Curie Actions.

Acknowledgements This research benefited from the support of FMJH program PGMO and from the support of EDF-Thalès-Orange.



© Evripidis Bampis, Bruno Escoffier, Kevin Schewior, and Alexandre Teiller;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 11; pp. 11:1–11:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In a classical combinatorial optimization setting, given an instance of a problem one needs to find a good feasible solution. However, in many situations, the data may evolve over time and one has to solve a sequence of instances. The natural approach of solving every instance independently may induce a significant transition cost, for instance for moving a system from one state to another. This cost may represent, e.g., the cost of turning on/off the servers in a data center [17, 8, 4, 1], the cost of changing the quality level in video streaming [16], or the cost for turning on/off nuclear plants [24]. Gupta et al. [15] and Eisenstat et al. [13] proposed a *multistage* model where given a time horizon $t = 1, 2, \dots, T$, the input is a sequence of instances I_1, I_2, \dots, I_T , (one for each time step), and the goal is to find a sequence of solutions S_1, S_2, \dots, S_T (one for each time step) reaching a trade-off between the quality of the solutions in each time step and the stability/similarity of the solutions in consecutive time steps. The addition of the transition cost makes some classic combinatorial optimization problems much harder. This is the case for instance for the minimum weighted perfect matching problem in the *off-line* case where the whole sequence of instances is known in advance. While the one-step problem is polynomially-time solvable, the multistage problem becomes hard to approximate even for bipartite graphs and for only two time steps [5, 15].

In this work, we focus on the *online* case, where at time t no knowledge is available for instances at times $t + 1, \dots, T$. When it is not possible to handle the online case, we turn our attention to the *k-lookahead* case, where at time t the instances at times $t + 1, \dots, t + k$ are also known. This case is of interest since in some applications like in dynamic capacity planning in data centers, the forecasts of future demands may be very helpful [18, 19]. Our goal is to measure the impact of the lack of knowledge of the future on the quality and the stability of the returned solutions. Indeed, our algorithms are limited in their knowledge of the sequence of instances. Given that the number of time steps is given, we compute the competitive ratio of the algorithm after time step T : As we focus on maximization problems, we say that a (deterministic) algorithm is (strictly) α -competitive (with competitive ratio α) if its value is at least $\frac{1}{\alpha}$ times the optimal value on all instances.

As it is usual in the online setting, we consider no limitations in the computational resources available. This implies that at every time step t , where instance I_t is known, we assume the existence of an oracle able to compute the optimal solution for that time step. Notice also that our lower bounds do not rely on any complexity assumptions. Some recent results are already known for the online multistage model [6, 15], however all these results are obtained for specific problems. In this work, we study multistage variants of a broad family of maximization problems. The family of optimization problems that we consider is the following.

► **Definition 1** (Subset Maximization Problems). *A Subset Maximization problem \mathcal{P} is a combinatorial optimization problem whose instances $I = (N, p, \mathcal{F})$ consist of*

- *A ground set N ;*
- *A set $\mathcal{F} \subseteq 2^N$ of feasible solutions such that $\emptyset \in \mathcal{F}$;*
- *A non-negative weight $p(S)$ for every $S \in \mathcal{F}$.*

The goal is to find $S^ \in \mathcal{F}$ such that $p(S^*) = \max\{p(S) : S \in \mathcal{F}\}$.*

We will consider that the empty set is always feasible, ensuring that the feasible set of solutions is non empty. This is a very general class of problems, including the maximization *Subset Selection* problems studied by Pruhs and Woeginger in [23] (they only considered linear objective functions). It contains for instance graph problems where N is the set of

vertices (as in any maximization induced subgraph problem verifying some property) or the set of edges (as in matching problems). It also contains classical set problems (knapsack, maximum 3-dimensional matching, . . .), and more generally 0-1 linear programs (with non negative profits in the objective function).

Given a problem in the previous class, we are interested in its multistage version [15, 13]. The stability over time of a solution sequence is classically captured by considering a transition cost when a modification is made in the solution. Here, dealing with maximization problems, we will consider a transition *bonus* B for taking into account the similarity of two consecutive solutions. In what follows, we will use the term *object* to denote an element of N (so an object can be a vertex of a graph, or an edge, . . ., depending on the underlying problem).

► **Definition 2** (Multistage Subset Maximization Problems). *In a Multistage Subset Maximization problem \mathcal{P} , we are given*

- a number of steps $T \in \mathbb{N}$, a set N of n objects;
- for any $t \in T$, an instance I_t of the optimization problem. We will denote:
 - p_t the objective (profit) function at time t
 - $\mathcal{F}_t \in 2^N$ the set of feasible solutions at time t
- $B \in \mathbb{R}^+$ a given transition profit.
- the value of a solution sequence $\mathcal{S} = (S_1, \dots, S_T)$ is

$$f(\mathcal{S}) = \sum_{t=1}^T p_t(S_t) + \sum_{t=1}^{T-1} b(S_t, S_{t+1})$$

where $b(S_t, S_{t+1})$ is the transition bonus for the solution between time steps t and $t + 1$. We will use the term *profit* for $p_t(S_t)$, *bonus* for the transition bonus $b(S_t, S_{t+1})$, and *value* of a solution \mathcal{S} for $f(\mathcal{S})$;

- the goal is to determine a solution sequence of maximum value.

The fact that T is known may be regarded as rather uncommon the field of online algorithms. At the end of Subsection 1.2, we relate it to our results and justify it.

There are two natural ways to define the transition bonus. We will see that these two ways of measuring the stability induce some differences in the competitive ratios one can get.

► **Definition 3** (Types of transition bonus). *If S_t and S_{t+1} denote, respectively, the solutions for time steps t and $t + 1$, then we can define the transition bonus as:*

- Intersection Bonus: B times $|S_t \cap S_{t+1}|$: in this case the bonus is proportional to the number of objects in the solution at time t that remain in it at time $t + 1$.
- Hamming Bonus: B times $|S_t \cap S_{t+1}| + |\overline{S_t} \cap \overline{S_{t+1}}|$. Here we get the bonus for each object for which the decision (to be in the solution or not) is the same between time steps t and $t + 1$. In other words, the bonus is proportional to $|N|$ minus the number of modifications (Hamming distance) in the solutions.

Note that by scaling profits (dividing them by B), we can arbitrarily fix $B = 1$. So from now on, we assume $B = 1$.

In this article, we will consider two possible ways for the data to evolve.

► **Definition 4** (Types of data evolution). ■ **Static Set of Feasible Solutions (SSFS):** *only profits may change over time, so the structure of feasible solutions remains the same: $\mathcal{F}_t = \mathcal{F}$ for all t .*

- **General Evolution (GE):** *any modification in the input sequence is possible. Both the profits and the set of feasible solutions may change over time. In this latter model, for knapsack, profits and weights of object (and the capacity of the bag) may change over time; for maximum independent set edges in the graph may change, . . .*

1.1 Related Work

A series of papers consider the online or semi-online settings, where the input changes over time and the algorithm has to modify (re-optimize) the solution by making as few changes as possible (see [3, 9, 12, 14, 20, 21] and the references therein). The multistage model considered in this paper has been introduced in Eisenstat et al. [13] and Gupta et al. [15]. Eisenstat et al. [13] studied the multistage version of facility location problems. They proposed a logarithmic approximation algorithm. An et al. [2] obtained constant factor approximation algorithms for some related problems. Gupta et al. [15] studied the MULTISTAGE MAINTENANCE MATROID problem for both the offline and the online settings. They presented a logarithmic approximation algorithm for this problem, which includes as a special case a natural multistage version of SPANNING TREE. They also considered the online version of the problem and they provide an efficient randomized competitive algorithm against any oblivious adversary. The same paper also introduced the study of the MULTISTAGE MINIMUM PERFECT MATCHING problem for which they proved that it is hard to approximate even for a constant number of stages. Bampis et al. [5] improved this negative result by showing that the problem is hard to approximate even for bipartite graphs and for the case of two time steps. When the edge costs are metric within every time step they proved that the problem remains APX-hard even for two time steps. They also showed that the maximization version of the problem admits a constant factor approximation algorithm, but is APX-hard. Olver et al. [22] studied a multistage version of the MINIMUM LINEAR ARRANGEMENT problem, which is related to a variant of the LIST UPDATE problem [25], and provided a logarithmic lower bound for the online version and a polylogarithmic upper bound for the offline version.

The MULTISTAGE MAX-MIN FAIR ALLOCATION problem has been studied in the offline and the online settings in [6]. This problem corresponds to a multistage variant of the SANTA KLAUS problem. For the off-line setting, the authors showed that the multistage version of the problem is much harder than the static one. They provided constant factor approximation algorithms for the off-line setting. For the online setting they proposed a constant competitive ratio for SSFS-type evolving instances and they proved that it is not possible to find an online algorithm with bounded competitive ratio for GE-type evolving instances. Finally, they showed that in the 1-lookahead case, where at time step t we know the instance of time step $t + 1$, it is possible to get a constant approximation ratio.

Buchbinder et al. [11] and Buchbinder, Chen and Naor [10] considered a multistage model and they studied the relation between the online learning and competitive analysis frameworks, mostly for fractional optimization problems.

1.2 Summary of Results and Overview

The contribution of our paper is a framework for online multistage maximization problems (comprising different models), a characterization of those models in which a constant competitive ratio is achievable, and almost tight upper and lower bounds on the best-possible competitive ratio for these models. The focus here is on deterministic algorithms.

We increase the complexity of the considered models over the course of the paper. We start with the arguably simplest model: Considering a static set of feasible solutions clearly restricts the general model of evolution; while such a straightforward comparison between the Hamming and intersection bonus is not possible, the Hamming bonus seems simpler in that, compared to the intersection model, there are (somewhat comparable) extra terms added on the profit of both the algorithm and the optimum. As we show in Subsection 2.1,

■ **Table 1** Our bounds on the best-possible competitive ratio c^* for the different models. The Landau symbol is with respect to $T \rightarrow \infty$.

	static set of feasible solutions	general evolution
Hamming bonus	$2 - o(1) \leq c^* \leq 2$	$1 + \sqrt{2} \leq c^* \leq 3 + o(1)$
	Theorems 6 and 5	Theorems 12 and 10
Intersection bonus	$2 \leq c^* \leq 2 + o(1)$	$c^* = \infty$
	Theorems 9 and 8	$c^* = 4$ for 1-lookahead Theorems 14, 16, and 15

there is indeed a simple 2-competitive algorithm: At each time t , it greedily chooses the set S_t that either maximizes the transition bonus w.r.t. S_{t-1} (that is, choosing $S_t = S_{t-1}$, which is possible in this model) or maximizes the value $p_t(S_t)$. We complement this observation with a matching lower bound only involving two time steps.

We then toggle the transition-bonus model and the data-evolution model separately and show that constant competitive ratios can still be achieved. First, in Subsection 2.2, we consider intersection bonus. We show that, *after* modifying the profits (internally) to make larger solutions more profitable, a $(2 + 1/(T - 1))$ -competitive algorithm can be achieved by a greedy approach again. We also give an (almost matching) lower bound of 2 again. Next, we toggle the evolution model. In Subsection 3.1, we adapt the greedy algorithm from Subsection 2.1 by reweighting to obtain a $(3 + 1/(T - 1))$ -competitive algorithm using a more complicated analysis. We complement this result with a lower bound of $1 + \sqrt{2}$.

In Subsection 3.2, we finally consider the general-evolution model with intersection bonus, where we give a simple lower bound showing that a constant-competitive ratio is not achievable. This lower bound relies on forbidding to choose any item in the second step that the algorithm chose in the first step. We circumnavigate such issues by allowing the algorithm a lookahead of one step and present a 4-competitive algorithm for that setting. A similar phase transition has been observed for a related problem [6], but our algorithm, based on a doubling approach, is different. We also give a matching lower bound of 4 on the competitive ratio of any algorithm in the same setting. We summarize all results described thus far in Table 1.

We note that the lower bounds mentioned for the Hamming model are only shown for a specific fixed number of time steps, and that in general there is no trivial way of extending these bounds to a larger number of time steps. One may however argue that the large- T regime is in fact the interesting one for both practical applications and in theory, the latter because the effect of having a first time step without bonus vanishes. At the end of the respective sections, we therefore give asymptotical lower bounds of $3/2$ and roughly 1.696 for the cases of a static set of feasible solutions and general evolutions, respectively. These bounds are non-trivial, but we do not know if they are tight.

It is plausible that the aforementioned upper bounds can be improved if extra assumptions on characteristics of the objective function and the sets of feasible solutions are made. In Subsubsection 3.1.2, we show that already very natural assumptions suffice: Assuming that at each time the feasible solutions are closed under taking subsets and the objective function is subadditive, we give a $(21/8 + o(1))$ -competitive algorithm for the model with a general evolution and Hamming bonus, improving the previous $(3 + o(1))$ -competitive ratio. Our lower bounds for general evolution and Hamming bonus in fact fulfill the extra assumptions.

We observe that all our algorithms except for the one discussed in Subsection 2.1 require that T is known in that their behavior in the last step is different from the behavior in the steps before. This assumption is crucial: In all these models, there are examples in which one can in the first time step choose either a small profit or a potentially large bonus not knowing if there is another timestep to realize the bonus. Such examples imply a superconstant lower bound on the competitive ratio in these models. This justifies our assumption that T is known.

In Section 4, we summarize our results and mention directions for future research that we consider interesting.

Due to space constraints, some proofs only appear in the full version [7].

2 Model of a Static Set of Feasible Solutions

We consider here the model of evolution where only profits change over time: $\mathcal{F}_t = \mathcal{F}$ for any t . We first consider the Hamming bonus model and show a simple 2-competitive algorithm. We will then show that a (asymptotic) competitive ratio of 2 can also be achieved in the intersection bonus model using a more involved algorithm. In both cases, this ratio 2 is shown to be (asymptotically) optimal.

2.1 Hamming-Bonus Model

► **Theorem 5.** *In the SSFS model with Hamming bonus, there is a 2-competitive algorithm.*

Proof. We consider the very simple following algorithm. At each time step t , the algorithm computes an optimal solution S_t^* with associated profit $p_t(S_t^*)$. At $t = 1$ we fix $S_1 = S_1^*$. For $t > 1$, if $p_t(S_t^*) > n$ then fix $S_t = S_t^*$, otherwise fix $S_t = S_{t-1}^*$ (which is possible thanks to the fact that the set of feasible solutions does not change).

Let f^* be the optimal value. Since any solution sequence gets profit at most $p_t(S_t^*)$ at time t , and bonus at most n between two consecutive time steps, we get $f^* \leq \sum_{t=1}^T p(S_t^*) + n(T-1)$.

By construction, at time $t > 1$, either the algorithm gets profit $p_t(S_t^*)$ when $p_t(S_t^*) > n$, or bonus (from $t-1$) n when $n \geq p_t(S_t^*)$. So in any case the algorithm gets profit plus bonus at least $\frac{p_t(S_t^*) + n}{2}$. At time 1 it gets profit at least $p_1(S_1^*)$. So

$$f(S_1, \dots, S_T) \geq p_1(S_1^*) + \sum_{t=2}^T \frac{p_t(S_t^*)}{2} + \frac{n(T-1)}{2} \geq \frac{f^*}{2},$$

which completes the proof. ◀

► **Theorem 6.** *Consider the SSFS model with Hamming bonus. For any $\epsilon > 0$, there is no $(2 - \epsilon)$ -competitive algorithm, even if there are only 2 time steps.*

Proof. We consider a set $N = \{1, 2, \dots, n\}$ of $n = 1 + \lceil \frac{1}{\epsilon} \rceil$ objects, $T = 2$ time steps, and an additive profit function. There are three feasible solutions: $S^0 = \emptyset$, $S^1 = \{1\}$ and $S^2 = \{2, \dots, n\}$. At $t = 1$, all the profits are 0. Let us consider an online algorithm A. We consider the three possibilities for the algorithm at time 1:

- At time 1, A chooses S^0 : at time 2 we give profit 1 to all objects. If A takes no object at time 2, it gets profit 0 and bonus n . If it takes S^1 , it gets profit 1 and bonus $n - 1$. If it takes S^2 , it gets profit $n - 1$ and bonus 1, so in any case the computed solution has value n . The solution consisting of taking S^2 at both time steps has profit $n - 1$ and bonus n , so value $2n - 1$.

- At time 1, A chooses S^1 : at time 2 we give profit 0 to object 1, and profit 1 to all other objects. Then, if the algorithm takes S^0 (resp, S^1 , S^2), at time 2 its gets value $n - 1$ (resp, n , $n - 1$) while the solution consisting of taking S^2 at both time steps has value $2n - 1$.
 - At time 1, A chooses S^2 : at time 2 we give profit n to object 1, and 0 to all other objects. Then if the algorithm takes S^0 (resp, S^1 , S^2) at time 2 its gets value 1 (resp, n , n), while the solution consisting of taking S^1 at both time steps has value $2n$.
- In any case, the ratio is at least $\frac{2n-1}{n} = 2 - \frac{1}{n} > 2 - \epsilon$. ◀

We complement this lower bound with an asymptotical result for large T ; the proof is provided in the full version.

► **Theorem 7.** *Consider the SSFS model with Hamming bonus. For every $\epsilon > 0$, there is a T_ϵ such that, for each number of time steps $T \geq T_\epsilon$, there is no $(3/2 - \epsilon)$ -competitive algorithm.*

2.2 Intersection-Bonus Model

In the intersection-bonus model things get harder since an optimal solution S_t^* may be of small size and then gives very small (potential) bonus for the next step. As a matter of fact, the algorithm of the previous section has unbounded competitive ratio in this case: take a large number n of objects, $\mathcal{F} = 2^N$, and at time 1 all objects have profit 0 up to one which has profit ϵ . The algorithm will take this object (instead of taking $n - 1$ objects of profit 0) and then potentially get bonus at most 1 instead of $n - 1$.

Thus we shall put an incentive for the algorithm to take solutions of large size, in order to have a chance to get a large bonus. We define the following algorithm called **MP-Algo** (for Modified Profit algorithm). Informally, at each time step t , the algorithm computes an optimal solution with a modified objective function p'_t . These modifications take into account (1) the objects taken at time $t - 1$ (2) an incentive to take a lot of objects. Formally, **MP-Algo** works as follows:

1. At $t = 1$: let $p'_1(S) = p_1(S) + |S|$. Choose S_1 as an optimal solution for the problem with modified profits p'_1 .
2. For t from 2 to $T - 1$: let $p'_t(S) = p_t(S) + |S \cap S_{t-1}| + |S|$. Choose S_t as an optimal solution for the problem with modified profit function p'_t .
3. At $t = T$: let $p'_T(S) = p_T(S) + |S \cap S_{T-1}|$. Choose S_T as an optimal solution with modified profit function p'_T .

The cases $t = 1$ and $t = T$ are specific since there is no previous solution for $t = 1$, and no future solution for $t = T$.

► **Theorem 8.** *In the SSFS model with intersection bonus, **MP-Algo** is $\left(\frac{2}{1-1/(T-1)}\right)$ -competitive.*

Proof. Let $(\hat{S}_1, \dots, \hat{S}_T)$ be an optimal sequence. Since S_t is optimal with respect to p'_t , for $t = 2, \dots, T - 1$ we have:

$$p'_t(S_t) = p_t(S_t) + |S_t \cap S_{t-1}| + |S_t| \geq p'_t(\hat{S}_t) \geq p_t(\hat{S}_t) + |\hat{S}_t|. \quad (1)$$

Since S_{t-1} is also a feasible solution at time t , we have:

$$p'_t(S_t) = p_t(S_t) + |S_t \cap S_{t-1}| + |S_t| \geq p_t(S_{t-1}) \geq 2|S_{t-1}|. \quad (2)$$

Similarly, at $t = T$ $p'_T(S) = p_T(S) + |S \cap S_{T-1}|$ so

$$p_T(S_T) + |S_T \cap S_{T-1}| \geq p_T(\hat{S}_T), \quad (3)$$

$$p_T(S_T) + |S_T \cap S_{T-1}| \geq |S_{T-1}|. \quad (4)$$

At $t = 1$, $p'_t(S) = p_t(S) + |S|$, so

$$p_1(S_1) + |S_1| \geq p_1(\hat{S}_1) + |\hat{S}_1|. \quad (5)$$

Now, note that $|S_t \cap S_{t-1}|$ is the transition bonus of the computed solution between $t - 1$ and t . By summing Equation (1) for $t = 2, \dots, T - 1$, Equation (3) and Equation (5), we deduce:

$$f(S_1, \dots, S_T) + \sum_{t=1}^{T-1} |S_t| \geq \sum_{t=1}^T p_t(\hat{S}_t) + \sum_{t=1}^{T-1} |\hat{S}_t|.$$

Since in the optimal sequence the transition bonus between time t and $t + 1$ is at most $|\hat{S}_t|$, we get:

$$f(S_1, \dots, S_T) + \sum_{t=1}^{T-1} |S_t| \geq f(\hat{S}_1, \dots, \hat{S}_T). \quad (6)$$

Now we sum Equation (2) for $t = 2, \dots, T - 1$ and Equation (4):

$$f(S_1, \dots, S_T) + \sum_{t=2}^{T-1} |S_t| \geq 2 \sum_{t=2}^{T-1} |S_{t-1}| + |S_{T-1}|.$$

From this we easily derive:

$$f(S_1, \dots, S_T) \geq \sum_{t=2}^{T-2} |S_t|. \quad (7)$$

By summing Equations (6) and (7) we have $2f(S_1, \dots, S_T) \geq f(\hat{S}_1, \dots, \hat{S}_T) - |S_{T-1}|$. The competitive ratio follows from the fact that $f(\hat{S}_1, \dots, \hat{S}_T) \geq (T - 1)|S_{T-1}|$ (since S_{T-1} is feasible for all time steps). ◀

We note that competitive ratio 2 can be derived with a similar analysis when the number of time steps is 2 or 3. In the full version, we show a matching asymptotical lower bound.

► **Theorem 9.** *Consider the SSFS model with intersection bonus. For any $\epsilon > 0$ and number of time steps $T = \lceil 1/\epsilon \rceil$, there is no $(2 - \epsilon)$ -competitive algorithm.*

3 Model of General Evolution

We consider in this section that the set of feasible solutions may evolve over time. We will show that in the Hamming bonus model, we can still get constant competitive ratios, though ratios slightly worse than in the case where only profits could change over time. Then, we will tackle the intersection bonus model, showing that no constant competitive ratio can be achieved. However, with only 1-lookahead we can get a constant competitive ratio.

3.1 Hamming-Bonus Model

In this section we consider the Hamming bonus model. We first show in Section 3.1.1 that there exists a $\left(3 + \frac{1}{T-1}\right)$ -competitive algorithm. Interestingly, we then show in Section 3.1.2 that a slight assumption on the problem structure allows to improve the competitive ratio. More precisely, we achieve a $21/8$ (asymptotic) competitive ratio if we assume that the objective function is subadditive (so including the additive case) and that a subset of a feasible solution is feasible. These assumptions are satisfied by all the problems mentioned in introduction. We finally consider lower bounds in Section 3.1.3.

3.1.1 General Case

We adapt the idea of the 2-competitive algorithm working for the Hamming bonus model for a static set of feasible solutions (Section 2.1) to the current setting where the set of feasible solutions may change. Let us consider the following algorithm **BestOrNothing**: at each time step t , **BestOrNothing** computes an optimal solution S_t^* with associated profit $p_t(S_t^*)$ and compares it to 2 times the maximum potential bonus, i.e to $2n$. It chooses S_t^* if the associated profit is at least $2n$, otherwise it chooses $S_t = \emptyset$. A slight modification is applied for the last step T .

In the full version, we define the algorithm formally and prove an upper bound on the competitive ratio achieved by this algorithm.

► **Theorem 10.** *In the GE model with Hamming bonus, **BestOrNothing** is $\left(3 + \frac{1}{T-1}\right)$ -competitive.*

3.1.2 Improvement for Sub-additivity and Subset Feasibility

In this section we assume that the problem have the following two properties:

- *subset feasibility*: at any time step, every subset of a feasible solution is feasible.
- *sub-additivity*: for any disjoint S, S' , any t , $p_t(S \cup S') \leq p_t(S) + p_t(S')$.

Note that this implies that, if a feasible set X is partitioned into (disjoint) subsets X_1, \dots, X_h , then X_1, \dots, X_h are feasible and $p_t(X) \leq \sum_i p_t(X_i)$.

We exploit this property to devise algorithms where we partition the set of objects and solve the problems on subinstances. As a first idea, let us partition the set of objects into 3 sets A, B, C of size (roughly) $n/3$; consider the algorithm which at every time step t computes the best solutions S_t^A, S_t^B, S_t^C on each subinstance on A, B and C , and chooses S_t as the one of maximum profit between these 3 solutions. By sub-additivity and subset feasibility, the algorithm gets profit at least $1/3$ of the optimal profit at each time step. Dealing with bonuses, at each time step the algorithm chooses a solution included either in A , or in B , or in C so, for any $t < T$, at least one set among A, B and C is not chosen neither at time t nor at time $t + 1$, and the algorithm gets transition bonus at least $n/3$. Hence, the algorithm is 3-competitive.

We now improve the previous algorithm. The basic idea is to remark that if for two consecutive time steps $t, t + 1$ the solution S_t and S_{t+1} are taken in the same subset, say A , then the bonus is (at least) $2n/3$ instead of $n/3$. Roughly speaking, we can hope for a ratio better than $1/3$ for the bonus. Then the algorithm makes a trade-off at every time step: if the profit is very high then it will take a solution maximizing the profit, otherwise it will do (nearly) the same as previously. More formally, let us consider the algorithm **3-Part**. We first assume that n is a multiple of 3. A parameter $x \in \mathbb{R}_+$ will be defined later.

1. Partition N in three subsets A, B, C of size $n/3$.
2. For $t \in \{1, \dots, T\}$: compute a solution S_t^* maximizing $p_t(S)$
 - Case (1): If $p_t(S_t^*) \geq xn$: define $S_t = S_t^*$
 - Otherwise ($p_t(S_t^*) \leq xn$): compute solutions with optimal profit S_t^A, S_t^B, S_t^C included in A, B and C . Let a_t, b_t and c_t the respective profits.
 - Case (2): if $t \geq 2$ and Case (1) did not occur at $t - 1$, do:
 - If $S_{t-1} \subseteq A$ (resp. $S_{t-1} \subseteq B, S_{t-1} \subseteq C$), compute $\max\{a_t + 2n/3, b_t + n/3, c_t + n/3\}$ (resp. $\max\{a_t + n/3, b_t + 2n/3, c_t + n/3\}, \max\{a_t + n/3, b_t + n/3, c_t + 2n/3\}$) and define S_t as S_t^A, S_t^B or S_t^C accordingly.
 - Case (3) ($t = 1$ or Case (1) occurred at $t - 1$) do:
 - * Define S_t as the solution with maximum profit among S_t^A, S_t^B, S_t^C .

11:10 Online Multistage Subset Maximization Problems

If N is not a multiple of 3, we add one or two dummy objects that are in no feasible solutions (at any step).

In the full version, we prove an upper bound on the competitive ratio of this algorithm.

► **Theorem 11.** *Consider the GE model with Hamming bonus. Under the assumption of subset feasibility and sub-additivity, 3-Part is $(21/8 + O(1/T + 1/n))$ -competitive.*

3.1.3 Lower Bounds

We complement the algorithmic results with a lower bound for two time steps and an asymptotical one, which we both prove in the full version. Interestingly, these bounds are also valid for the latter restricted setting with subset feasibility and sub-additivity.

► **Theorem 12.** *Consider the GE model with Hamming bonus and $T = 2$ time steps. For any $\epsilon > 0$, there is no $(1 + \sqrt{2} - \epsilon)$ -competitive algorithm.*

► **Theorem 13.** *Consider the GE model with Hamming bonus. For every $\epsilon > 0$, there is a T_ϵ such that, for each number of time steps $T \geq T_\epsilon$, there is no $(\alpha - \epsilon)$ -competitive algorithm where $\alpha = \frac{6 \cdot \sqrt[3]{9 + \sqrt{87}}}{\sqrt[3]{6 \cdot (9 + \sqrt{87})^2 - \sqrt[3]{36}}} \approx 1.696$.*

3.2 Intersection-Bonus Model

We now look at the general-evolution model with intersection bonus. This model is different from the ones considered before: We first give a simple lower bound showing that there is no constant-competitive algorithm.

► **Theorem 14.** *In the GE model with intersection bonus, there is no c -competitive algorithm for any constant c .*

Proof. We consider an instance with no profit. Let $T = 2$, $N = \{1, 2\}$, and $\mathcal{F}_1 = \{\emptyset, \{1\}, \{2\}\}$, that is, there are two items, and at time 1 it is only forbidden to take both of them. Assume w.l.o.g. that the algorithm does not pick item 2 at time 1. Then picking item 1 becomes infeasible at time 2 while picking item 2 remains feasible. Then the algorithm achieves 0 profit and bonus while the optimum can achieve a bonus of 1. ◀

Note that in this model, by adding dummy time steps giving no bonus and no profit, the previous lower bound extends to any number of time steps. This lower bound motivates considering the 1-lookahead model: at time t , besides I_t , the algorithm knows the instance I_{t+1} . It shall decide the feasible solution chosen at time t . We consider an algorithm based on the following idea: at some time step t , the algorithm computes an optimal sequence of 2 solutions $(S_{t,1}^*, S_{t,2}^*)$ of value z_t^* for the subproblem defined on time steps t and $t + 1$. Suppose it fixes $S_t = S_{t,1}^*$. Then, at time $t + 1$, it computes $(S_{t+1,1}^*, S_{t+1,2}^*)$ of value z_{t+1}^* . Depending on the values z_t^* and z_{t+1}^* , it will either choose to set $S_{t+1} = S_{t,2}^*$, confirming its choice at t (getting in this case value z_t^* for sure between time t and $t + 1$), or change its mind and set $S_{t+1} = S_{t+1,1}^*$ (possibly no value got yet, but a value z_{t+1}^* if it confirms this choice at $t + 2$). When a choice is confirmed ($S_t = S_{t,1}^*$ and $S_{t+1} = S_{t,2}^*$), then the algorithm starts a new sequence (fix $S_{t+2} = S_{t+2,1}^*, \dots$).

More formally, let $(S_{t,1}^*, S_{t,2}^*)$ be an optimal solution of the subproblem defined on time steps t and $t + 1$, and denote z_t^* its value (including profits and bonus between time t and $t + 1$). To avoid unnecessary subcases, we consider at time T $(S_{T,1}^*, S_{T,2}^*)$ where $S_{T,2} = \emptyset$ and z_T^* is the profit of the optimal solution for the single time step T , $S_{T,1}^*$. Then consider the algorithm **Balance** which:

1. At time $t = 1$ compute $(S_{1,1}^*, S_{1,2}^*)$ and fix $S_1 = S_{1,1}^*$.
2. For $t = 2$ to T : compute $(S_{t,1}^*, S_{t,2}^*)$.
 - Case (1): If at $t - 1$ the algorithm chose S_{t-1} equal to $S_{t-2,2}^*$ (i.e., Case (3) occurred), then fix $S_t = S_{t,1}^*$.
 - Case (2): Otherwise, if $z_t^* > 2z_{t-1}^*$, then fix $S_t = S_{t,1}^*$.
 - Case (3): Otherwise fix $S_t = S_{t-1,2}^*$.

► **Theorem 15.** *In the GE model with intersection bonus and 1-lookahead, Balance is a 4-competitive algorithm.*

Proof. Let V be the set of time steps in which Case (3) occurred. In the proof, intuitively we partition the time period into periods which end at some time $t \in V$, and prove the claimed ratio in each of these sub-periods.

Formally, let u, v ($u < v$) be two time steps in V such that $w \notin V$ for any $u < w < v$. Note that since Case (3) occurred at time u , Case (1) occurred at $u + 1$, so $u \neq v - 1$, and Case (2) occurred at time $u + 2, \dots, v - 1$. So $z_t^* > 2z_{t-1}^*$ for $t = u + 2, \dots, v - 1$. By an easy recurrence, this means that, for all $t \in \{u + 1, \dots, v - 1\}$, we have $z_t^* < z_{v-1}^*/2^{v-1-t}$. By taking the sum, we get $\sum_{t=u+1}^{v-1} z_t^* < 2z_{v-1}^*$. Since Case (3) occurred at v , $z_v^* \leq 2z_{v-1}^*$. Finally:

$$\sum_{t=u+1}^v z_t^* \leq 4z_{v-1}^*.$$

Now, at each time v for which case (3) occurred, we choose $S_v = S_{v-1,2}^*$. As previously said, Case (3) did not occur at $v - 1$, so we choose $S_{v-1} = S_{v-1,1}^*$. Then the algorithm gets value at least z_{v-1}^* for these two time steps. In other words $f(S_1, \dots, S_T) \geq \sum_{v \in V} z_{v-1}^*$. Consider first the case where $T \in V$ (case (3) occurred at time T). Then we get a partition of the time steps into subintervals ending in $v \in V$. So

$$\sum_{t=1}^T z_t^* \leq 4 \sum_{v \in V} z_{v-1}^* \leq 4f(S_1, \dots, S_T).$$

Let $(\hat{S}_1, \dots, \hat{S}_T)$ be an optimal solution. We have $p_t(\hat{S}_t) + p_{t+1}(\hat{S}_{t+1}) + b(\hat{S}_t, \hat{S}_{t+1}) \leq z_t^*$. So $f(\hat{S}_1, \dots, \hat{S}_T) \leq \sum_{t=1}^{T-1} z_t^*$, and:

$$f(\hat{S}_1, \dots, \hat{S}_T) \leq \sum_{t=1}^T z_t^* \leq 4f(S_1, \dots, S_T).$$

Note that this is overestimated, each $p_t(\hat{S}_t)$ appears two times in the sum.

Now, if $T \notin V$, then $T - 1 \in V$: indeed, Case (2) cannot occur at time T (since $z_T^* \leq z_{T-1}^*$). So we have in this case:

$$\sum_{t=1}^{T-1} z_t^* \leq 4 \sum_{v \in V} z_{v-1}^* \leq 4f(S_1, \dots, S_T).$$

But again since $f(\hat{S}_1, \dots, \hat{S}_T) \leq \sum_{t=1}^{T-1} z_t^*$, we have

$$f(\hat{S}_1, \dots, \hat{S}_T) \leq \sum_{t=1}^{T-1} z_t^* \leq 4f(S_1, \dots, S_T).$$

This completes the proof. ◀

In the full version, we prove a matching lower bound. The idea of the proof of the matching lower bound is as follows: As can be seen from the proof of Theorem 15, the estimate on the profit has slack for the 4-competitive algorithm. We give a construction in which there is no profit and in which the bonus when not “committing” to the solution from the previous time step is geometrically increasing over time; otherwise the bonus is 0. As it turns out, however, when the factor is 2 in each time step, we cannot show a lower bound of 4 in case the algorithm does not commit until the last time step. Interestingly, if we use the minimum factor to show a lower bound of $4 - \epsilon$ in case the algorithm commits at any time step but the last, we can find a large-enough time horizon such that, in case the algorithm commit only in the last time step, we can also show a lower bound of $4 - \epsilon$.

► **Theorem 16.** *Consider the GE model with intersection bonus. For any $\epsilon > 0$, there is a T_ϵ such that, for each number of time steps $T \geq T_\epsilon$, there is no $4 - \epsilon$ competitive ratio.*

4 Conclusion

In this paper, we have developed techniques for online multistage subset maximization problems and thereby settled the achievable competitive ratios in the various settings almost exactly. Disregarding asymptotically vanishing terms in the upper bounds, what remains open is the exact ratio in the general-evolution setting with Hamming bonus (shown to be between $1 + \sqrt{2}$ and 3 in this paper) and exact bounds for the models with Hamming bonus when $T \rightarrow \infty$. Furthermore, it is plausible that the ratios can be improved for (classes of) more specific problems.

We emphasize that we have focussed on deterministic algorithms in this work. Indeed, some of our bounds can be improved by randomization (assuming an oblivious adversary):

- In the general-evolution model with Hamming bonus assuming sub-additivity and subset feasibility, there is a simple randomized $(2 + o(1))$ -competitive algorithm (along the lines of the algorithms in Subsubsection 3.1.2): Initially partition N uniformly at random into two equal-sized sets (up to possibly one item) A and B . At each time, select the optimal solution restricted to A . Again, the algorithm is $(2 + o(1))$ -competitive separately on both profit and bonus.
- While the strong lower bound without lookahead in the general-evolution model with intersection bonus still holds, we can get a simple 2-competitive algorithm for lookahead 1: Initially flip a coin to interpret the instance as a sequence of length-2 instances either starting at time 1 or 2. Thanks to lookahead 1, the length-2 instances can all be solved optimally. The total value of all these length-2 instances adds up to at least the optimal value, and the expected value obtained by the algorithm is half of that.

While we believe that we have treated various of the most natural ways of defining value in multistage subset maximization problems, other ways can be thought of, to some of which our results extend. For instance, Theorem 5 also works for time-dependent or object-dependent bonus without major modifications (whereas, e.g., Theorem 8 does not).

We have not worried about computational complexity in this work (and therefore neither about the representation of the set of feasible solutions); indeed, often we use an oracle providing the optimal solution to instances of a potentially hard problem. However, we mention that, if only an approximation algorithm to the problem at hand was known, we would be able to obtain similar online algorithms whose competitive ratio would depend on the approximation guarantee of the approximation algorithm.

References

- 1 Susanne Albers and Jens Quedenfeld. Optimal Algorithms for Right-Sizing Data Centers. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 363–372, 2018.
- 2 Hyung-Chan An, Ashkan Norouzi-Fard, and Ola Svensson. Dynamic Facility Location via Exponential Clocks. *ACM Trans. Algorithms*, 13(2):21:1–21:20, 2017.
- 3 Barbara M. Anthony and Anupam Gupta. Infrastructure Leasing Problems. In *Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 424–438, 2007.
- 4 Antonios Antoniadis and Kevin Schewior. A Tight Lower Bound for Online Convex Optimization with Switching Costs. In *Workshop on Approximation and Online Algorithms (WAOA)*, pages 164–175, 2017.
- 5 Evripidis Bampis, Bruno Escoffier, Michael Lampis, and Vangelis Th. Paschos. Multistage Matchings. In *Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 7:1–7:13, 2018.
- 6 Evripidis Bampis, Bruno Escoffier, and Sasa Mladenovic. Fair Resource Allocation Over Time. In *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 766–773, 2018.
- 7 Evripidis Bampis, Bruno Escoffier, Kevin Schewior, and Alexandre Teiller. Online Multistage Subset Maximization Problems. *CoRR*, abs/1905.04162, 2019. [arXiv:1905.04162](https://arxiv.org/abs/1905.04162).
- 8 Nikhil Bansal, Anupam Gupta, Ravishankar Krishnaswamy, Kirk Pruhs, Kevin Schewior, and Clifford Stein. A 2-Competitive Algorithm For Online Convex Optimization With Switching Costs. In *Workshop on Approximation, Randomization, and Combinatorial Optimization Algorithms and Techniques (APPROX/RANDOM)*, pages 96–109, 2015.
- 9 Nicolas K. Blanchard and Nicolas Schabanel. Dynamic Sum-Radii Clustering. In *International Conference and Workshops on Algorithms and Computation (WALCOM)*, pages 30–41, 2017.
- 10 Niv Buchbinder, Shahar Chen, and Joseph Naor. Competitive Analysis via Regularization. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 436–444, 2014.
- 11 Niv Buchbinder, Shahar Chen, Joseph Naor, and Ohad Shamir. Unified Algorithms for Online Learning and Competitive Analysis. *Math. Oper. Res.*, 41(2):612–625, 2016. doi:10.1287/moor.2015.0742.
- 12 Edith Cohen, Graham Cormode, Nick G. Duffield, and Carsten Lund. On the Tradeoff between Stability and Fit. *ACM Trans. Algorithms*, 13(1):7:1–7:24, 2016. doi:10.1145/2963103.
- 13 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.
- 14 Albert Gu, Anupam Gupta, and Amit Kumar. The Power of Deferral: Maintaining a Constant-Competitive Steiner Tree Online. *SIAM J. Comput.*, 45(1):1–28, 2016. doi:10.1137/140955276.
- 15 Anupam Gupta, Kunal Talwar, and Udi Wieder. Changing Bases: Multistage Optimization for Matroids and Matchings. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 563–575, 2014.
- 16 Vinay Joseph and Gustavo de Veciana. Jointly optimizing multi-user rate adaptation for video transport over wireless systems: Mean-fairness-variability tradeoffs. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 567–575, 2012.
- 17 Minghong Lin, Adam Wierman, Lachlan L. H. Andrew, and Eno Thereska. Dynamic Right-Sizing for Power-Proportional Data Centers. *IEEE/ACM Trans. Netw.*, 21(5):1378–1391, 2013.
- 18 Minghong Lin, Adam Wierman, Lachlan L. H. Andrew, and Eno Thereska. Dynamic Right-Sizing for Power-Proportional Data Centers. *IEEE/ACM Trans. Netw.*, 21(5):1378–1391, 2013.

11:14 Online Multistage Subset Maximization Problems

- 19 Zhenhua Liu, Iris Liu, Steven H. Low, and Adam Wierman. Pricing data center demand response. In *ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 111–123, 2014.
- 20 Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. The Power of Recourse for Online MST and TSP. *SIAM J. Comput.*, 45(3):859–880, 2016. doi:10.1137/130917703.
- 21 Chandrashekhar Nagarajan and David P Williamson. Offline and online facility leasing. *Discrete Optimization*, 10(4):361–370, 2013.
- 22 Neil Olver, Kirk Pruhs, Rene Sitters, Kevin Schewior, and Leen Stougie. The Itinerant List-Update Problem. In *Workshop on Approximation and Online Algorithms (WAOA)*, pages 310–326, 2018.
- 23 Kirk Pruhs and Gerhard J. Woeginger. Approximation schemes for a class of subset selection problems. *Theor. Comput. Sci.*, 382(2):151–156, 2007.
- 24 Cécile Rottner. *Combinatorial Aspects of the Unit Commitment Problem*. PhD thesis, Sorbonne Université, 2018.
- 25 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized Efficiency of List Update and Paging Rules. *Commun. ACM*, 28(2):202–208, 1985.

A Constant Approximation for Colorful k -Center

Sayan Bandyapadhyay

Department of Computer Science, University of Iowa, Iowa City, IA, USA
sayan-bandyapadhyay@uiowa.edu

Tanmay Inamdar

Department of Computer Science, University of Iowa, Iowa City, IA, USA
tanmay-inamdar@uiowa.edu

Shreyas Pai

Department of Computer Science, University of Iowa, Iowa City, IA, USA
shreyas-pai@uiowa.edu

Kasturi Varadarajan

Department of Computer Science, University of Iowa, Iowa City, IA, USA
kasturi-varadarajan@uiowa.edu

Abstract

In this paper, we consider the colorful k -center problem, which is a generalization of the well-known k -center problem. Here, we are given red and blue points in a metric space, and a coverage requirement for each color. The goal is to find the smallest radius ρ , such that with k balls of radius ρ , the desired number of points of each color can be covered. We obtain a constant approximation for this problem in the Euclidean plane. We obtain this result by combining a “pseudo-approximation” algorithm that works in any metric space, and an approximation algorithm that works for a special class of instances in the plane. The latter algorithm uses a novel connection to a certain matching problem in graphs.

2012 ACM Subject Classification Theory of computation \rightarrow Facility location and clustering; Theory of computation \rightarrow Computational geometry

Keywords and phrases Colorful k -center, Euclidean Plane, LP Rounding, Outliers

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.12

Funding The first, second and fourth authors are partially supported by the National Science Foundation under Grant CCF-1615845.

1 Introduction

In the k -center problem, we are given a finite metric space (P, d) , where P is a set of n points, and $d : P \times P \rightarrow \mathbb{R}^+$ is the associated distance function. We are also given an integer $1 \leq k \leq n$. The goal is to find a subset $C \subseteq P$ of *centers*, where $|C| = k$, so as to minimize $\max_{p \in P} \min_{c \in C} d(p, c)$, the maximum distance of a point from its nearest center. Geometrically, we want to find the smallest radius ρ such that P can be covered by k balls of radius ρ (centered at points in P).

It is well known that k -center is NP-hard; furthermore, it is also NP-hard to approximate the optimal radius to within a $2 - \epsilon$ factor, for any $\epsilon > 0$. This is easily seen via a reduction from the minimum dominating set problem [16]. On the other hand, it is possible to obtain a tight approximation ratio of 2 [11, 15]. A simple greedy algorithm of [11] achieving this starts with C containing an arbitrary point in P ; in each of the subsequent $k - 1$ iterations, it finds a point $p \in P$ that maximizes $d(p, C) := \min_{c \in C} d(p, c)$, and adds that point to C .

In the k -center with outliers problem, we are given an additional parameter $1 \leq p \leq n$. The goal is to find the smallest radius ρ such that at least p points of P can be covered by k balls of radius ρ . Thus, we allow up to $n - p$ points to remain uncovered, and these



© Sayan Bandyapadhyay, Tanmay Inamdar, Shreyas Pai, and Kasturi Varadarajan;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 12; pp. 12:1–12:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

would be “outliers.” Intuitively, in comparison to k -center, an algorithm for solving this problem has to also figure out which p points to cover. Nevertheless, Charikar et al. [6], who introduced this problem, gave a simple 3-approximation using a greedy algorithm. Their algorithm guesses the optimal radius ρ ; then, for k iterations, it finds the ball of radius ρ that covers the maximum number of yet uncovered points, expands it by a factor of 3, and adds it to the solution. More recently, the approximation guarantee was improved to 2, using LP-rounding techniques [4, 13]. Note that this approximation guarantee is tight, in light of the $2 - \epsilon$ hardness result mentioned earlier.

We now introduce the *colorful k -center* problem, which is studied in this article. As in k -center, the input consists of a set P of n points in a metric space, and an integer k . Furthermore, we are given a partition $\{P_1, P_2, \dots, P_c\}$ of P into c *color classes*, and a coverage requirement $0 \leq t_i \leq |P_i|$ for each color class $1 \leq i \leq c$. The goal is to find the smallest radius ρ such that using k balls of radius ρ , centered at points of P , we can simultaneously cover at least t_i points from each class P_i . When we have only one color class, that is, when $c = 1$, we obtain the k -center with outliers problem. In much of this article, we focus on the case with two color classes, where $c = 2$. In this case, we call the colors *red* and *blue*; we denote P_1 and P_2 , respectively, by R (red points) and B (blue points), and denote the coverage requirements t_1 and t_2 , respectively, by r and b . The motivation for studying this problem is as follows. Each color class can be thought of as a certain demographic with a specific coverage requirement, which must be satisfied by the k balls chosen in the solution.

Even with two colors, the colorful k -center is quite challenging. The greedy algorithm for k -center with outliers [6] has no obvious generalization: with two color classes, what do we optimize when adding the next ball to our solution? The LP-based approaches for k -center with outliers do not generalize either – as we point out (in Example 5), the natural LP has an unbounded integrality gap.

Chakrabarty et al. [4] study a closely related problem called non-uniform k center. As input to this problem, we are given a set P of points in a metric space, λ distinct radii $r_1 > \dots > r_\lambda \geq 0$ and corresponding integers t_1, \dots, t_λ . The goal is to find the smallest *dilation* $\beta \geq 0$ such that P can be covered by a collection of balls formed by including, for each $1 \leq i \leq \lambda$, t_i balls of radius $\beta \cdot r_i$. When $\lambda = 1$, we get the regular k -center problem. When λ is unbounded, the problem is hard to approximate to within any constant factor; when $\lambda = 2$, one can get an $O(1)$ -approximation; and when λ is a constant greater than 2, it is open as to whether an $O(1)$ -approximation is possible [4].

As Chakrabarty et al. [4] observe, there is a close relationship between k -center with outliers and non-uniform k -center with $\lambda = 2$. In fact, it can be shown that the two problems are equivalent up to an $O(1)$ -approximation factor. While the relationship between the colorful k -center with c color classes and non-uniform k -center with $c + 1$ distinct radii is not known to be as close for $c \geq 2$, the study of the latter problem is one motivation that led us to the colorful k -center problem.

1.1 Other Related Work

The k -means and k -median are classic NP-hard clustering problems that are closely related to the k -center problem. Like the k -center problem, these problems have been extensively studied, resulting in different approaches guaranteeing constant factor approximations. More recently, the outlier versions of these problems were also studied; constant factor approximations were obtained for k -median with outliers [8, 19] and k -means with outliers [19]. A polynomial time bicriteria $(1 + \epsilon)$ -approximation using at most $k(1 + \epsilon)$ centers for any $\epsilon > 0$ is known in low dimensional Euclidean spaces, and metric spaces with constant doubling dimension [10].

Facility location with outliers, which is referred to as *Robust Facility Location*, is a generalization of the uncapacitated Facility Location problem; various constant approximations are known for the latter problem. The Robust Facility Location problem was introduced in [6], who gave a 3-approximation. The approximation guarantee was later improved by Jain et al. [18] to 2.

A colorful version of vertex cover is studied in [1], and colorful versions of the Set Cover and Facility Location-type problems were considered in [17]. In these problems, the cardinality of the cover (or its weight) shows up in the objective function, unlike in k -center, where the number of centers/balls k is a “hard restriction”. These problems therefore have a different flavor.

Finally, k -center and k -median have been generalized in an orthogonal direction, where there are additional constraints on the centers [12, 7, 5]. Again, the issues studied in these generalizations tend to be quite different from the ones we confront here.

1.2 Our Results

We study the colorful k -center problem when the number of colors is a constant. Our main result is a randomized polynomial time algorithm that, with high probability, outputs an $O(1)$ -approximation in the Euclidean plane. (As k -center is APX-hard even in the plane [9], we cannot hope for a PTAS.)

To describe our approach, we focus on the case with two colors. We first design a pseudo-approximation algorithm that outputs a 2-approximation, but with $k + 1$ centers instead of k . This result holds in any metric space, not just the Euclidean plane. To obtain it, we preprocess the solution to a natural LP-relaxation into a solution for a simpler LP, using ideas from [4, 13]. We then note that a basic feasible solution to the simple LP opens at most $k + 1$ centers fractionally. A pseudo-approximation for k -median with outliers was an important step in the recent work of Krishnaswamy et al. [19].

The pseudo-approximation allows us to reduce the colorful k -center problem to a special case where the balls in the optimal solution are separated – the distance between any two balls is much greater than their radii. Designing an $O(1)$ -approximation for this special case is challenging, even in the plane. For instance, partitioning into small subproblems by using a grid (such as in [14]) or other type of object does not work, because any such partitioning may intersect balls in the optimal solution, and we have a hard bound k on the number of balls allowed.

We solve separated instances in the plane by reducing to exact perfect matching on graphs. In this problem, we are given a graph in which each edge has a red weight and a blue weight, both non-negative integers. Given integers w_r and w_b , the goal is to determine if the graph has a perfect matching whose red and blue weights are, respectively, *exactly* w_r and w_b . This problem can be solved in randomized polynomial time, provided the weights are bounded by a polynomial in the input size; see [3, 21]. To our knowledge, this connection of geometric clustering and covering to exact matching is a novel one.

In its current form, the reduction from separated instances of colorful k -center to exact matching does not work even in \mathbb{R}^3 . Nevertheless, we are hopeful that this work will lead to an $O(1)$ -approximation to colorful k -center in dimensions 3 and higher, and indeed in any metric space.

Organization. We describe our pseudo-approximation in Section 2 and our $O(1)$ -approximation for the Euclidean plane in Section 3. In both these sections, we focus on the case where the number of colors is 2. We address the extension of the planar result to multiple color classes in Section 4.

Find a feasible solution (x, z) such that:	
$\sum_{i \in B(j, \rho)} x_i \geq z_j, \quad \forall j \in P$	(1)
$\sum_{i \in P} x_i \leq k,$	(2)
$\sum_{j \in R} z_j \geq r,$	(3)
$\sum_{j \in B} z_j \geq b,$	(4)
$z_j, x_i \in [0, 1], \quad \forall i, j \in P$	(5)

■ **Figure 1** The feasibility LP, parameterized by ρ .

2 Pseudo-approximation via LP Rounding

Recall that an instance of colorful k -center is a metric space $(P = R \sqcup B, d)$, where R, B are non-empty, disjoint sets of red and blue points respectively. We are also given red and blue coverage requirements $1 \leq r \leq |R|$ and $1 \leq b \leq |B|$, respectively. A solution (C, D) , where $C, D \subseteq P$, is said to be feasible, if (i) $|C| \leq k$, (ii) $|D \cap R| \geq r$, and (iii) $|D \cap B| \geq b$. The cost of a solution is defined as $\max_{j \in D} d(j, C)$. Any point in the set D is said to be *covered* by the solution. The goal of the colorful k -center problem is to find a feasible solution of the minimum cost.

In this section, we describe a pseudo-approximation algorithm for the colorful k -center problem. That is, we show how to find a solution of cost at most $2 \cdot OPT$ using at most $k + 1$ centers, where OPT is the cost of an optimal solution. This result is achieved in two steps. In the first step, we use the natural LP relaxation for the decision version of the colorful k -center problem, to partition the points in P into disjoint clusters using a simple greedy procedure. We also obtain a related fractional solution during this clustering procedure. We use this to show that a much simpler LP with a small number of constraints has a feasible solution, even though there may not exist an integral feasible solution using at most k centers. Nevertheless, we use the simplicity of the LP to show that there exists a solution using at most $k + 1$ centers.

Let ρ denote our “guess” for the optimal cost. Note that the optimal cost must be one of the $O(n^2)$ interpoint distances, therefore there are $O(n^2)$ choices for ρ . We state the feasibility LP, parameterized by ρ , in Figure 1. It is easy to see that an optimal solution satisfies all the constraints when the guess ρ is correct (i.e., when $\rho \geq OPT$). Therefore, henceforth, we assume that $\rho = OPT$.

Now, we find a feasible fractional solution (x', z') for this LP, and use it to show that a related, but a much simpler LP is feasible. To this end, we use the following “greedy clustering” procedure (see Algorithm 1), which also computes a modified LP solution (\tilde{x}, \tilde{z}) . Let P' denote the set of unclustered points, initialized to P . We also initialize S , the collection of cluster-centers, to the empty set. In each iteration, we find a point $j \in P'$ with the maximum z_j . We set $\tilde{z}_j = \tilde{x}_j \leftarrow \min\{1, \sum_{i \in B(j, \rho)} x'_i\}$ – note that the sum is over all points in the ball $B(j, \rho)$, as opposed to the points in $B(j, \rho) \cap P'$. Let C_j denote the set of unclustered points

Algorithm 1 Clustering Algorithm.

```

1:  $S \leftarrow \emptyset, P' \leftarrow P$ 
2: while  $P' \neq \emptyset$  do
3:    $j \in P'$  be a point with maximum  $z'_j$ ; let  $S \leftarrow S \cup \{j\}$ 
4:    $\tilde{x}_j \leftarrow \min\{1, \sum_{i \in B(j, \rho)} x'_i\}$ ;  $\tilde{z}_j \leftarrow \tilde{x}_j$ 
5:    $C_j \leftarrow B(j, 2\rho) \cap P'$ 
6:   For all  $j' \neq j \in C_j$ , set  $\tilde{x}_{j'} \leftarrow 0, \tilde{z}_{j'} \leftarrow \tilde{z}_j$ 
7:    $P' \leftarrow P' \setminus C_j$ 
8: end while
  
```

within distance 2ρ from j . We refer to the set C_j as a cluster. We set $\tilde{z}_{j'} \leftarrow \tilde{z}_j$ for all other points $j' \in C_j \setminus \{j\}$. Finally, we remove the points in C_j from P' and repeat this process until all points are clustered, i.e., P' becomes empty.

For any point $i \in S$, let $R_i := R \cap C_i$ and $B_i := B \cap C_i$ denote the sets of red and blue points in the “cluster” C_i respectively. Additionally, for any $i \in S$, let r_i and b_i denote the sizes of the sets R_i and B_i respectively. We start with a few simple observations that are immediate from the description of the procedure.

► **Observation 1.**

1. The “clusters” $\{C_i\}_{i \in S}$ partition the point set P . Therefore, $\{R_i\}_{i \in S}$ partition the red points R , and $\{B_i\}_{i \in S}$ partition the blue points B ,
2. For any two distinct $i, i' \in S$, $d(i, i') > 2\rho$,
3. For any $j \in P$, there is at most one $i \in S$, such that $d(i, j) \leq \rho$.

The following observation follows from the greedy nature of the clustering procedure.

► **Observation 2.** For any point $j_1 \in P$, let $j \in S$ be the point in S such that $j_1 \in C_j$. Then, $\tilde{z}_j = \tilde{z}_{j_1} \geq z'_{j_1}$

Proof. Notice that, $\tilde{z}_{j_1} = \tilde{z}_j = \min\{1, \sum_{i \in B(j, \rho)} x'_i\}$. There are two cases to consider.

$\tilde{z}_{j_1} = \tilde{z}_j = 1 \geq z'_{j_1}$, where the inequality follows from the constraint (5) of the LP.

Otherwise, $\tilde{z}_j = \sum_{i \in B(j, \rho)} x'_i$. In this case, $\tilde{z}_j = \sum_{i \in B(j, \rho)} x'_i \geq z'_j \geq z'_{j_1}$. Here, the first inequality follows from constraint (1) of the LP, and the second inequality follows from the fact that in the iteration when j_1 was removed from P' , $j \in P'$ was chosen to be a point with the maximum z' -value in line 3. ◀

The next two claims help us construct a feasible solution to a simplified LP, to be introduced later.

▷ **Claim 3.**

1. $\sum_{i \in S} r_i \tilde{x}_i \geq r$
2. $\sum_{i \in S} b_i \tilde{x}_i \geq b$

$$\begin{aligned}
 & \text{maximize} && \sum_{i \in S} r_i x_i \\
 & \text{subject to} && \sum_{i \in S} b_i x_i \geq b && (6) \\
 & && \sum_{i \in S} x_i \leq k, && (7) \\
 & && x_i \in [0, 1], \quad \forall i \in S && (8)
 \end{aligned}$$

■ **Figure 2** The Simplified LP.

Proof. We prove the first part of the claim – the second part is analogous.

$$\begin{aligned}
 \sum_{i \in S} r_i \tilde{x}_i &= \sum_{i \in S} |R_i| \cdot \tilde{x}_i \\
 &= \sum_{i \in S} \sum_{j' \in R_i} \tilde{z}_i && \text{(For any } i \in S, \tilde{x}_i = \tilde{z}_i \text{ by construction)} \\
 &\geq \sum_{i \in S} \sum_{j' \in R_i} z'_{j'} && \text{(For any } j' \in R_i \subseteq C_i, \tilde{z}_i \geq z'_{j'} \text{ from Observation 2)} \\
 &= \sum_{j \in R} z'_j && \text{(Property 1 of Observation 1)} \\
 &\geq r && \text{(By constraint (3))}
 \end{aligned}$$

◁

▷ **Claim 4.** $\sum_{i \in S} \tilde{x}_i \leq k$

Proof.

$$\begin{aligned}
 \sum_{i \in S} \tilde{x}_i &\leq \sum_{i \in S} \sum_{i' \in B(i, \rho)} x'_{i'} && (\tilde{x}_i \leq \sum_{i' \in B(i, \rho)} x'_{i'}) \\
 &\leq \sum_{i' \in P} x'_{i'} && \text{(From Property 3 of Observation 1)} \\
 &\leq k && \text{(By constraint (2))}
 \end{aligned}$$

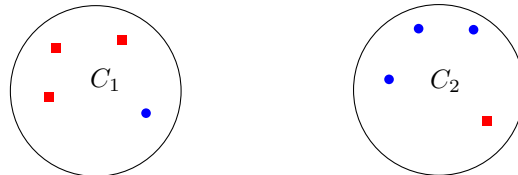
◁

2.1 A Simplified Problem

Recall that the clusters $\{C_j\}_{j \in S}$ are disjoint, and have radius 2ρ . Now, if we can find a collection of k clusters from this set, that cover at least r red points and b blue points, then this immediately leads to a 2-approximation. Unfortunately, it is not always possible to find such a collection. However, in the following, we show that we can find a collection of $k + 1$ clusters that satisfies the coverage requirements of both colors. The LP in Figure 2 is a relaxation of the problem of finding at most k clusters that satisfy the coverage requirements of both colors.

Note that Claims 3 and 4 imply that the fractional solution (\tilde{x}) constructed above is a feasible solution for this LP, and has objective value at least r . However, there may not exist a feasible integral solution that uses at most k clusters from the set $\{C_i\}_{i \in S}$. The following example illustrates this.

► **Example 5.** In the following figure, we have two clusters C_1 and C_2 . Red points are shown as boxes, whereas blue points are shown as dots. C_1 consists of 3 red points and 1 blue point, whereas C_2 contains 3 blue points and 1 red point. Suppose that $k = 1$ and the coverage requirements of each color class is 2. Now, assigning $x_1 = x_2 = 0.5$ yields a fractional solution that satisfies the coverage requirements of each color class. However, assuming that the distance between two clusters is very large compared to the radii, 2ρ , it can be seen that there is no feasible integral solution of cost at most a constant multiple of 2ρ .



Nevertheless, we show in the following that there exists a feasible solution that uses at most $k + 1$ clusters. First, we need the following classical result from linear algebra (see Lemma 2.1.4 in [20]).

► **Lemma 6** ([20]). *In any extreme point feasible solution (or equivalently, a basic feasible solution) to a linear program, the number of linearly independent tight constraints is equal to the number of variables.*

Furthermore, an extreme point optimal solution can be computed in polynomial time. Now, we find such an optimal solution (x^*) to the simplified LP. It follows from Claim 3 that its objective value is at least r . We prove the following lemma, which is a simple consequence of Lemma 6.

► **Lemma 7.** *The number of x^* -variables in an extreme point optimal solution that are strictly fractional, is at most 2. Therefore, the number of strictly positive variables is at most $k + 1$.*

Proof. Let $m := |S|$ denote the number of variables. From Lemma 6, it follows that the number of linearly independent tight constraints is equal to m . Note that, even though there are $2m + 2$ constraints in the LP, at most $m + 2$ constraints can be simultaneously tight. Now, even if constraints (6) and (7) are tight, it follows that number of tight constraints from (8) is at least $m - 2$. That is, the number of strictly fractional variables is at most 2.

If there are k variables that are equal to 1, then constraint (7) is tight, and there are no strictly fractional variables. Otherwise, the number of variables equal to 1, is at most $k - 1$. Along with at most 2 strictly fractional variables, the number of positive variables is at most $k + 1$. ◀

Note that the red and blue coverage can only increase while rounding up the fractional variables to 1. If there is exactly one fractional variable, we round it up, giving a 2-approximation using exactly k centers. Otherwise, let x_1^*, x_2^* be the two fractional variables, corresponding to clusters C_1, C_2 respectively. Note that, although the fractional solution satisfies both red and blue coverage, adding only C_1 or C_2 to the solution may not satisfy coverage demands for both colors – recall Example 5. Therefore, we include both clusters in the solution, resulting in a pseudo-approximation of cost at most 2ρ , using at most $k + 1$ centers. We summarize the result of this section in the following theorem.

► **Theorem 8.** *There exists a polynomial time algorithm to find a 2-approximation for the colorful k -center problem (with two colors) in any metric space, using at most $k + 1$ centers.*

12:8 A Constant Approximation for Colorful k -Center

This theorem generalizes readily to an arbitrary number of color classes $c \geq 2$. Having c color classes corresponds to having $c - 1$ constraints instead of constraint (6) in the corresponding Simplified LP. As we now have constraints corresponding to $c - 1$ color classes, we obtain the following lemma for the Simplified LP, the proof of which goes along the same lines as that of Lemma 7.

► **Lemma 9.** *The number of x^* -variables in an extreme point optimal solution that are strictly fractional, is at most c . Therefore, the number of strictly positive variables is at most $k + c - 1$.*

Therefore, if we round up these fractional variables, we get a pseudo-approximation of cost at most $2 \cdot \rho$, using at most $k + c - 1$ centers.

3 A Constant Approximation Algorithm in \mathbb{R}^2

In this section, we describe a constant approximation algorithm for the colorful k -center problem with two color classes, red and blue, where the set of points lies in the Euclidean plane. Recall that $P = R \sqcup B$ denotes the input set of n points, and r and b the red and blue coverage requirements. Our algorithm consists of two subroutines which we describe in the following two subsections. The two subroutines are intended to handle two different types of instances. In order to describe these instances we need the following definitions.

► **Definition 10.** *Let $\alpha > 0$ be a parameter. A set S of balls of radius ρ' is α -separated if the distance between the centers of every two balls in S is greater than $\alpha \cdot \rho'$. An instance of colorful k -center is α -separated if the set of balls in some optimal solution is α -separated.*

The first subroutine (described in Section 3.1) gives a $2(\alpha + 1)$ -approximation algorithm for instances that are not α -separated. The second subroutine (described in Section 3.2) gives a $(0.5\alpha + 2)$ -approximation algorithm for α -separated instances where $\alpha > 4/\gamma$ for some absolute constant $\gamma > 0$, which is defined below.

Therefore for a large enough constant α , we get a $\max\{2(\alpha + 1), 0.5\alpha + 2\}$ -approximation algorithm for the colorful k -center problem with two color classes. From the geometric arguments in Section 3.2 that determine α , it is apparent that taking $\alpha = 8 + \epsilon'$ for any $\epsilon' > 0$ is sufficient. Therefore, the approximation guarantee of our algorithm is $17 + \epsilon$ for any $\epsilon > 0$.

3.1 Handling Non-Separated Instances

If an instance is not α -separated, we can use the pseudo-approximation algorithm of Section 2 to immediately get a $2(\alpha + 1)$ -approximation algorithm. This is formalized in the following lemma. It is worth pointing out that this subroutine does not require the set of points to be in \mathbb{R}^2 – it works for any metric.

► **Lemma 11.** *If an instance is not α -separated then we get a $2(\alpha + 1)$ -approximate solution to the colorful k -center problem in polynomial time.*

Proof. Let the optimal radius be ρ . Since the instance is not α -separated, there are two balls in some optimal solution whose centers are within distance $\alpha \cdot \rho$ of each other. Let C_1 and C_2 be two such balls in the optimal solution. We replace C_1 with a ball of radius $(\alpha + 1)\rho$ centered at the same point as C_1 . This allows us to remove C_2 without violating any of the coverage requirements (since the new ball replacing C_1 covers all points originally covered by C_2). This means that there exists a feasible solution using $k - 1$ centers with cost

$(\alpha + 1)\rho$. Therefore, if we run the pseudo-approximation algorithm of the previous section with number of centers being $k - 1$, we will get a solution using at most k centers having cost $2(\alpha + 1)\rho$, which proves the lemma. ◀

3.2 Reduction of Separated Instances to Exact Perfect Matching

Let us assume that the instance we are given is α -separated for some $\alpha > 4/\gamma$ for some absolute constant $\gamma > 0$. From the proof of Lemma 12, it will be clear that taking $\alpha > 8$ is sufficient. The α -separability of the instance helps us design a $(0.5\alpha + 2)$ -approximation algorithm for this problem in \mathbb{R}^2 . We do this by reducing the problem to the *Exact Perfect Matching* problem [22, 2]. In the exact perfect matching problem we are given an edge-weighted graph $G = (V, E)$ and a target weight W . The goal is to find a perfect matching in G having weight *exactly* W . The result in [3, 21] gives a randomized pseudo-polynomial algorithm for exact perfect matching. In other words, their algorithm runs in polynomial time if the largest edge weight in the input graph is bounded by a polynomial in $|V|$. We now describe our reduction from the k -center problem to exact perfect matching.

We first assume that we have guessed correctly the radius ρ of an optimal solution to the given instance. We cover \mathbb{R}^2 with a grid of equilateral triangles of side length $\ell = 0.5\alpha\rho$. See Figure 3 for an illustration. Consider the following three lines through the origin: L_1 is the x -axis, L_2 has angle 60 degrees with L_1 , and L_3 has angle 120 degrees with L_1 . The grid of side length ℓ can be formally defined as the arrangement of the collection of lines parallel to L_1 , L_2 , and L_3 , with two adjacent parallel lines having distance $\sqrt{3}\ell/2$. We can interpret the grid as an infinite graph where the edges are the sides of each atomic triangle in the grid and the vertices are the intersection points of the edges.

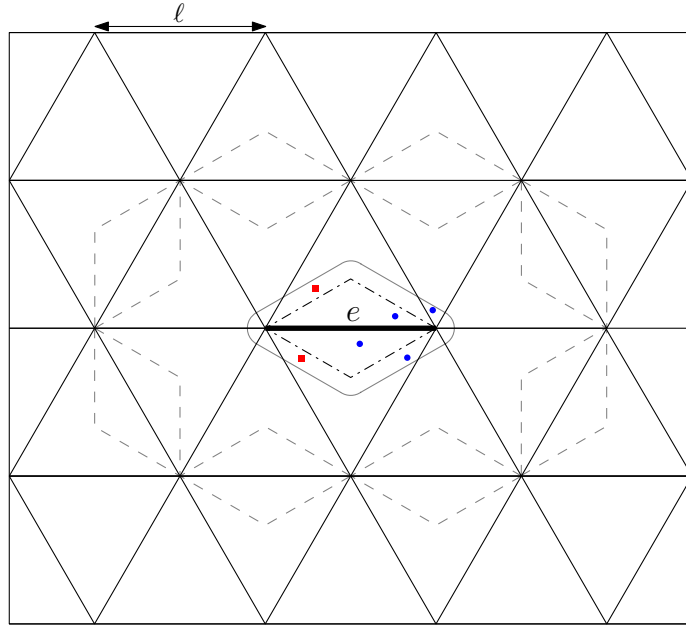
Define the *Voronoi region of an edge e* as the set of points that are at least as close to e as any other edge. It is easy to see that the Voronoi region of e is a rhombus whose four end points are the two end points of e and the centroids of the two triangles sharing the edge e . We are interested in the *extended Voronoi region of an edge e* which is the Minkowski sum of the Voronoi region of e and a ball of radius ρ centered at the origin.

► **Lemma 12.** *The extended Voronoi regions of two edges e and e' intersect each other iff e and e' share a common vertex.*

Proof. The reverse direction of the lemma is trivial, so we focus on the forward direction. Take two edges e and e' that are not incident on a common vertex. Note that their Voronoi regions do not intersect (see Figure 3). Thus, there is an absolute constant $\gamma > 0$ such that the distance between the closest pair of points, one in each region, is at least $\gamma \cdot \ell = \gamma \cdot 0.5\alpha\rho$. Therefore, if $\alpha > 4/\gamma$, the distance between the two Voronoi regions is greater than 2ρ .¹ Thus, the extended Voronoi regions of e and e' do not intersect because each Voronoi region “expands” by only an additive factor of ρ . ◀

For each edge define $P(e)$ as the set of points in P that lie in the extended Voronoi region of e . Let $G' = (V', E')$ be the graph where E' is the set of all grid edges such that $P(e) \neq \emptyset$, and V' is the set consisting of the endpoints of edges in E' . Define $N = 2 \max\{n, |V'|\}$, note that $N = O(n)$ because each point can belong to at most six triangles. The weight of an edge $e \in E'$ is set to $w_e = N^2 \cdot b_e + r_e$ where b_e and r_e are the number of blue and red points respectively in $P(e)$.

¹ With simple geometric calculations, one can show that $\gamma = 1/2$, which implies that $\alpha > 8$.



■ **Figure 3** Here, we show a triangular grid of side $\ell = 0.5\alpha\rho$. We also show the Voronoi region of the edge e (the area inside the dash-dotted rhombus around e), and the extended Voronoi region of e (the area inside the lightly shaded object). We also show the Voronoi regions of the “neighboring edges” that do not share a vertex with e as dashed rhombuses. Notice that the Voronoi regions (resp. extended Voronoi regions, not shown for simplicity) of the neighboring edges do not intersect with the Voronoi region (resp. extended Voronoi region) of e . We use this fact in the proof of Lemma 12. Notice that, the extended Voronoi region of e contains 2 red points and 4 blue points. Therefore, we will set the weight of the edge to be $4 \cdot N^2 + 2$.

► **Lemma 13.** *There exists a matching in G' with k edges and with weight exactly $b' \cdot N^2 + r'$ for some $|B| \geq b' \geq b$ and $|R| \geq r' \geq r$.*

Proof. Let S^* be the set of balls in an optimal solution that is α -separated. For each ball $C_i \in S^*$, $1 \leq i \leq k$, let e_i be any edge (from the infinite grid) such that the center of C_i lies in the Voronoi region of e_i . Note that the extended Voronoi region of e_i contains C_i , and thus $e_i \in E'$. We claim that if $i \neq j$, then (a) $e_i \neq e_j$, and (b) e_i and e_j are not incident on a common vertex of V' . If either (a) or (b) does not hold, the distance between the centers of C_i and C_j is at most $0.5\alpha\rho + 0.5\alpha\rho = \alpha\rho$, which violates the fact that the instance is α -separated. Therefore the set $M = \{e_i | 1 \leq i \leq k\}$ is a matching of k edges in G .

If we set b' and r' to be the total number of blue and red points in the extended Voronoi regions of the k edges, then we have a matching of weight $b' \cdot N^2 + r'$ in G . Since the extended Voronoi regions cover all the points covered by S^* , we have $b' \geq b$ and $r' \geq r$ which finishes the proof. ◀

We now define a graph G such that perfect matchings in G correspond to matchings with k edges in G' . To construct $G = (V, E)$, we start with G' and add $|V'| - 2k$ auxiliary vertices that are connected to all vertices of G' with edges of weight zero. We can do this because Lemma 13 implies that $|V'| \geq 2k$. Now G obeys the following property that is easy to prove.

► **Observation 14.** *Any matching in G' of weight W having exactly k edges can be extended to a perfect matching in G of weight exactly W . Conversely, from any perfect matching in G with weight exactly W , we can obtain a matching in G' with k edges and weight exactly W .*

Using Lemma 13 and Observation 14 we get the following Corollary.

► **Corollary 15.** *There exists a perfect matching in G of weight exactly $b' \cdot N^2 + r'$ for some $|B| \geq b' \geq b$ and $|R| \geq r' \geq r$.*

We now show how to go from a perfect matching in G to a cover for the input instance.

► **Lemma 16.** *If G has a perfect matching of weight exactly $W = b'N^2 + r'$, with $0 \leq b' \leq |B|$ and $0 \leq r' \leq |R|$, then we can place exactly k balls of radius $(0.5\alpha + 2)\rho$ that cover at least r' red points and at least b' blue points.*

Proof. Suppose G has a perfect matching M of weight W . Using Observation 14, we recover a matching M' in G' with k edges and weight W . For $e \in M'$, let $b_e = |P(e) \cap B|$ and $r_e = |P(e) \cap R|$. Thus, the weight w_e for each edge e is $b_e N^2 + r_e$. It follows that

$$\left(\sum_{e \in M'} b_e \right) N^2 + \left(\sum_{e \in M'} r_e \right) = W = b' N^2 + r'.$$

By our choice of N , both r' and $\sum_{e \in M'} r_e$ are strictly less than N^2 . It follows that $\sum_{e \in M'} r_e = r' = W \bmod N^2$, and furthermore $\sum_{e \in M'} b_e = b'$. Furthermore, since M' is a matching, by Lemma 12, the extended Voronoi regions of the edges in M' do not intersect. Thus, $\cup_{e \in M'} P(e)$ contains exactly r' red points and b' blue points.

For each $e \in M'$ such that $P(e) \neq \emptyset$, we place a ball of radius $(0.5\alpha + 2)\rho$ centered at any point in $P(e)$. These at most k balls cover all points in $\cup_{e \in M'} P(e)$. This is because the distance between any two points in $P(e)$ for an edge e is at most $(0.5\alpha + 2)\rho$. This finishes the proof of the lemma. ◀

We are now ready to state our algorithm for our α -separated instance of colorful k -center, assuming that we know the optimal radius ρ . We construct the graph G and check if it has a perfect matching of weight exactly $W = b' \cdot N^2 + r'$, for each $b \leq b' \leq |B|$ and $r \leq r' \leq |R|$. For the check, we invoke the algorithm of [3]. Corollary 15 ensures that for at least one of these guesses for W , an exact perfect matching does exist, and the algorithm of [3] returns it.² Once we find an exact perfect matching solution for any of these weight values, we convert it into a $(0.5\alpha + 2)$ -approximate solution by invoking the algorithm of Lemma 16. This completes our algorithm.

As we do not know the optimal radius for the given input instance, we will run the above algorithm for each choice of ρ from the set of $O(n^2)$ interpoint distances induced by P . Fix a particular choice of ρ . If there exists a feasible solution with balls of radius ρ that is α -separated, then the above algorithm will return a feasible solution with cost at most $(0.5\alpha + 2) \cdot \rho$. If there is no feasible solution with balls of radius ρ that is α -separated, then the above algorithm may not return a feasible cover.

Overall, we simply return the minimum cost feasible solution that we encounter after applying the above algorithm for all choices of ρ . This is a $(0.5\alpha + 2)$ -approximation for α -separated instances.

► **Remark 17.** We note that the algorithm described here generalizes to any constant number of color classes, in the following sense. Suppose we have an input instance for which there exists an α -separated solution of cost ρ' , for some ρ' . In particular, we do not assume that ρ'

² The algorithm of [3] is actually a randomized Monte-Carlo algorithm, and thus has an error probability that can be made arbitrarily small. For the sake of exposition, we ignore this eventuality of error, except in the statement of our final result.

12:12 A Constant Approximation for Colorful k -Center

is the optimal cost for the input instance. Then the algorithm computes a feasible solution of cost at most $(0.5\alpha + 2) \cdot \rho'$. Note that in this case, for any edge e of the graph G' , the weight of each edge will be of the form $\sum_{i \geq 0} P_i(e) N^{2i}$ where $P_i(e)$ is the number of points of color class i in the extended Voronoi region of that edge.

3.3 Combining Separated and Non-Separated Cases

We do not know if a given instance of colorful k -center is α -separated or not. Therefore, we do not know which subroutine to use. So we run both the subroutines and return the solution with the lowest cost. We conclude with our main result.

► **Theorem 18.** *There is a randomized polynomial time algorithm that, given any instance of colorful k -center in the plane with two colors, outputs, with high probability, an $O(1)$ -approximate solution.*

4 Multiple Color Classes in the Plane

So far we have only considered two color classes for the sake of keeping the exposition simple. We now sketch an extension to the colorful k -center problem in the plane with c color classes for any integer constant c .

For a given instance in the plane, fix the optimal set of balls S^* , and assume it has cost ρ . Let S be any maximal α -separated subset of S^* . That is, for each pair of balls in S , the distance between the centers is greater than $\alpha \cdot \rho$, whereas for every ball $B \in S^* \setminus S$, there is a ball $B' \in S$, such that the distance between the centers of B and B' is at most $\alpha \cdot \rho$. Let $\bar{S} = S^* \setminus S$. We consider two cases:

Case 1: $|\bar{S}| \geq c - 1$

Notice that if we expand each ball in S to have radius $(\alpha + 1) \cdot \rho$, and discard the balls in \bar{S} , we obtain a feasible solution with at most $k - (c - 1)$ balls and cost $(\alpha + 1) \cdot \rho$. For this case, running the pseudo-approximation algorithm on the original input, but with the number of centers set to $k - (c - 1)$, will give us a solution with k balls and cost $2(\alpha + 1) \cdot \rho$.

Case 2: $|\bar{S}| \leq c - 2$

We guess the balls in \bar{S} . Because $|\bar{S}| \leq c - 2$, and there are $O(n^2)$ choices for ρ , we have $O(n^c)$ possibilities to guess from. After guessing \bar{S} , we remove the points covered by \bar{S} , and decrease the coverage requirement for each color class by the number of points of that color class covered by \bar{S} . For this residual instance, S is an α -separated solution with $k - |\bar{S}|$ balls and cost ρ . If we run the algorithm of Section 3.2 on this residual instance, allowing $k - |\bar{S}|$ centers, we obtain a cover with $k - |\bar{S}|$ balls and cost at most $(0.5\alpha + 2) \cdot \rho$. We output the union of this cover and \bar{S} . This is a feasible solution to the original problem with k balls and cost $O(\rho)$.

The overall running time of the algorithm is $n^{O(c)}$.

5 Open Problems

One question raised by our work is whether an $O(1)$ -approximation for colorful k -center (with a constant number of colors) can be obtained in \mathbb{R}^d for $d \geq 3$, or indeed in an arbitrary metric space. Another question is whether one can obtain an $O(1)$ -approximation for non-uniform k -center in the plane (with a constant number of distinct radii).

References

- 1 Suman K Bera, Shalmoli Gupta, Amit Kumar, and Sambuddha Roy. Approximation algorithms for the partition vertex cover problem. *Theoretical Computer Science*, 555:2–8, 2014.
- 2 André Berger, Vincenzo Bonifaci, Fabrizio Grandoni, and Guido Schäfer. Budgeted matching and budgeted matroid intersection via the gasoline puzzle. *Mathematical Programming*, 128(1-2):355–372, 2011.
- 3 Paolo M. Camerini, Giulia Galbiati, and Francesco Maffioli. Random pseudo-polynomial algorithms for exact matroid problems. *Journal of Algorithms*, 13(2):258–273, 1992.
- 4 Deeparnab Chakrabarty, Prachi Goyal, and Ravishankar Krishnaswamy. The Non-Uniform k-Center Problem. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
- 5 Deeparnab Chakrabarty and Maryam Negahbani. Generalized Center Problems with Outliers. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- 6 Moses Charikar, Samir Khuller, David M Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 642–651. Society for Industrial and Applied Mathematics, 2001.
- 7 Danny Z Chen, Jian Li, Hongyu Liang, and Haitao Wang. Matroid and knapsack center problems. *Algorithmica*, 75(1):27–52, 2016.
- 8 Ke Chen. A constant factor approximation algorithm for k-median clustering with outliers. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 826–835. Citeseer, 2008.
- 9 Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 434–444. ACM, 1988.
- 10 Zachary Friggstad, Kamyar Khodamoradi, Mohsen Rezapour, and Mohammad R Salavatipour. Approximation schemes for clustering with outliers. *ACM Transactions on Algorithms (TALG)*, 15(2):26, 2019.
- 11 Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
- 12 MohammadTaghi Hajiaghayi, Rohit Khandekar, and Guy Kortsarz. Budgeted red-blue median and its generalizations. In *European Symposium on Algorithms*, pages 314–325. Springer, 2010.
- 13 David G Harris, Thomas Pensyl, Aravind Srinivasan, and Khoa Trinh. A Lottery Model for Center-Type Problems with Outliers. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
- 14 Dorit S Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM (JACM)*, 32(1):130–136, 1985.
- 15 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.
- 16 Wen-Lian Hsu and George L Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3):209–215, 1979.
- 17 Tanmay Inamdar and Kasturi R. Varadarajan. On the Partition Set Cover Problem. *CoRR*, abs/1809.06506, 2018. [arXiv:1809.06506](https://arxiv.org/abs/1809.06506).
- 18 Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM (JACM)*, 50(6):795–824, 2003.
- 19 Ravishankar Krishnaswamy, Shi Li, and Sai Sandeep. Constant approximation for k-median and k-means with outliers via iterative rounding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 646–659. ACM, 2018.

12:14 A Constant Approximation for Colorful k -Center

- 20 Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization*, volume 46. Cambridge University Press, 2011.
- 21 Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.
- 22 Christos H Papadimitriou and Mihalis Yannakakis. The complexity of restricted spanning tree problems. *Journal of the ACM (JACM)*, 29(2):285–309, 1982.


Parametrized Complexity of Expansion Height

Ulrich Bauer 

Department of Mathematics, Technical University of Munich (TUM),
Boltzmannstr. 3, 85748 Garching b. München, Germany
ulrich.bauer@tum.de

Abhishek Rathod 

Department of Mathematics, Technical University of Munich (TUM),
Boltzmannstr. 3, 85748 Garching b. München, Germany
abhishek.rathod@tum.de

Jonathan Spreer 

School of Mathematics and Statistics, The University of Sydney, NSW 2006 Australia
jonathan.spreer@sydney.edu.au

Abstract

Deciding whether two simplicial complexes are homotopy equivalent is a fundamental problem in topology, which is famously undecidable. There exists a combinatorial refinement of this concept, called simple-homotopy equivalence: two simplicial complexes are of the same simple-homotopy type if they can be transformed into each other by a sequence of two basic homotopy equivalences, an elementary collapse and its inverse, an elementary expansion. In this article we consider the following related problem: given a 2-dimensional simplicial complex, is there a simple-homotopy equivalence to a 1-dimensional simplicial complex using at most p expansions? We show that the problem, which we call the *erasability expansion height*, is $\mathbf{W[P]}$ -complete in the natural parameter p .

2012 ACM Subject Classification Mathematics of computing \rightarrow Algebraic topology; Theory of computation \rightarrow Problems, reductions and completeness; Theory of computation \rightarrow W hierarchy

Keywords and phrases Simple-homotopy theory, simple-homotopy type, parametrized complexity theory, simplicial complexes, (modified) dunce hat

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.13

Funding This research has been supported by the DFG Collaborative Research Center SFB/TRR 109 “Discretization in Geometry and Dynamics”.

Jonathan Spreer: The author is partially supported by grant EVF-2015-230 of the Einstein Foundation Berlin. The authors acknowledges support by The University of Sydney, where parts of this work were finished.

Acknowledgements We want to thank João Paixão for very helpful discussions.

1 Introduction

Homotopy theory lies at the heart of algebraic topology. In an attempt to make the concept of homotopy equivalence more amenable to combinatorial methods, Whitehead developed what turned out to be a combinatorial refinement of the theory, called *simple-homotopy theory*. Simple-homotopy theory considers sequences of elementary homotopy equivalences defined on simplicial complexes (or, more generally, CW complexes): an *elementary collapse*, which takes a face of a complex contained only in a single proper coface and removes both faces, and its inverse operation, called an *elementary expansion*. Two simplicial complexes are then said to be of the same simple-homotopy type if one can be transformed into the other by a sequence of elementary collapses and expansions. Complexes of the same simple-homotopy type are homotopy equivalent, but the converse is not always true [18], the obstruction being an element of the *Whitehead group* of the fundamental group. However, Whitehead



© Ulrich Bauer, Abhishek Rathod, and Jonathan Spreer;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 13; pp. 13:1–13:15



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

proved that all homotopy-equivalent complexes with a trivial fundamental group are in fact of the same simple-homotopy type [19], and thus in this particular case the notions of simple-homotopy and homotopy coincide.

A presentation of the fundamental group can be read off from a two-dimensional complex such that the presentation is balanced and describes the trivial group if and only if this complex is contractible [12]. Since the decidability of the triviality problem for balanced presentations is open [4], the same is also true for the decidability of contractibility of 2-complexes. Hence, the decidability of the existence of a simple-homotopy equivalence from a 2-complex to a point is also open. In contrast, the problem of deciding whether a given complex has trivial fundamental group is famously undecidable already for 2-complexes through its connection to the word problem, see, for instance, [5]. It follows that sequences of elementary collapses and expansions proving simple-homotopy equivalence between a 2-complex and a point can be expected to be long, if not unbounded. Nonetheless, understanding these sequences offers a great reward: the statement that any contractible 2-complex contracts to a point using only expansions up to dimension three is equivalent to a weaker variant [13, p. 34–35] of the Andrews–Curtis conjecture [1, 20].

In this article, motivated by the aforementioned problems, we investigate the computational (parametrized) complexity of a number of variants of the problem of deciding contractibility. More precisely, we focus on the problem of deciding whether a given 2-complex admits a simple-homotopy to a 1-complex using at most p expansions, called ERASIBILITY EXPANSION HEIGHT. In addition, we consider a variant, called ORDERED ERASIBILITY EXPANSION HEIGHT, which requires that all expansions come at the very beginning of the sequence. It is worth noting that ERASIBILITY EXPANSION HEIGHT and ORDERED ERASIBILITY EXPANSION HEIGHT are equivalent for CW complexes for which one can readily swap the order of expansions and collapses [13, p. 34]. However, for simplicial complexes, the ordered and unordered expansion heights may differ.

In Section 5.1, we prove that ERASIBILITY EXPANSION HEIGHT is $\mathbf{W}[\mathbf{P}]$ -hard, see Theorem 8. The proof uses a reduction from AXIOM SET. The same reduction also establishes $\mathbf{W}[\mathbf{P}]$ -hardness of ORDERED ERASIBILITY EXPANSION HEIGHT. Note that a reduction from AXIOM SET is also used by the third author and others in [6] to establish $\mathbf{W}[\mathbf{P}]$ -hardness of a parametrized version of OPTIMAL MORSE MATCHING. However, unlike in [6], the use of combinatorial and topological properties of the dunce hat is a key ingredient of the reduction used in this paper. In particular, there is only one gadget in the reduction – a subdivision of the so-called *modified dunce hat* [3], see Figure 2. In this sense the techniques used in this paper are also related to recent work by the first and second author in [3], where they show hardness of approximation for some Morse matching problems.

In Section 5.2, we show that ERASIBILITY EXPANSION HEIGHT and ORDERED ERASIBILITY EXPANSION HEIGHT are both in $\mathbf{W}[\mathbf{P}]$, and hence also $\mathbf{W}[\mathbf{P}]$ -complete, see Theorems 8, 13 and 14. Both results rest on the key observation that a 2-complex is erasable if and only if greedily collapsing triangles yields a 1-dimensional complex (Proposition 11), as shown by Tancer [17, Proposition 5].

In Section 6 we show that, as a consequence of the above reduction, the problem of deciding whether a 2-complex can be shown to be simple-homotopy equivalent to a 1-complex using only 3-dimensional expansions, called ERASIBILITY 3-EXPANSION HEIGHT, is \mathbf{NP} -complete, see Theorem 17.

2 Definitions and Preliminaries

2.1 Simplicial complexes

A (finite) *abstract simplicial complex* is a collection K of subsets of a finite ground set V such that if τ is an element of K , and σ is a nonempty subset of τ , then σ is an element of K . The ground set V is called the *set of vertices* of K . Since simplicial complexes are determined by their facets, we sometimes present simplicial complexes by listing their facets. A *subcomplex* of K is a subset $L \subseteq K$ which is itself a simplicial complex. Given a subset $W \subseteq V$ of the vertices of K , the *induced subcomplex on W* consists of all simplices of K that are subsets of W .

The elements of K are referred to as its *faces*. The *dimension of a face* is defined to be its cardinality minus one, and the *dimension of K* equals the largest dimension of its faces. For brevity, we sometimes refer to a d -dimensional simplicial complex as a *d -complex* and to a d -dimensional face as a *d -face*. The 0-, 1-, and 2-faces of a d -complex K are called its *vertices*, *edges*, and *triangles* respectively. Faces of K which are not properly contained in any other face are called *facets*. An $(m - 1)$ -face $\sigma \in K$ which is contained in exactly one m -face $\tau \in K$ is called *free*.

The *star* of a vertex v of complex K , written $\text{star}_K(v)$, is the subcomplex consisting of all faces of K containing v , together with their faces. If a map $\phi : V \rightarrow W$ between the vertex sets of two simplicial complexes K and L , respectively, sends every simplex $\sigma \in K$ to a simplex $\phi(\sigma) \in L$, then the induced map $f : K \rightarrow L, \sigma \mapsto \phi(\sigma)$, is said to be *simplicial*.

2.2 Simple-homotopy

We introduce the basic notions of simple-homotopy used in the present paper. The general concept of simple-homotopy can be understood independently from the notion of homotopy. In this sense this article aims to be self-contained. For further reading on homotopy theory we refer to [11].

In short, a simple-homotopy equivalence is a refinement of a homotopy equivalence. It can be described purely combinatorially with the help of the following definition.

► **Definition 1** (Elementary collapses and expansions). *Let K_0 be a simplicial complex, and let $\tau, \sigma \in K_0$ be an m -face and an $(m - 1)$ -face respectively such that $\sigma \subset \tau$, and σ is free in K_0 .*

We say that $K_1 = K_0 \setminus \{\tau, \sigma\}$ arises from K_0 by an elementary collapse of dimension m or elementary m -collapse, denoted by $K_0 \searrow K_1$. Its inverse, the operation $K_0 = K_1 \cup \{\tau, \sigma\}$ is called an elementary expansion of dimension m or elementary m -expansion, written $K_0 \nearrow K_1$. If the complex is implicit from context, we denote elementary collapses by \searrow_σ^τ and elementary expansions by \nearrow_σ^τ . An elementary collapse or an elementary expansion is sometimes referred to as an elementary move, or simply a move.

If there exists a sequence of elementary collapses turning a complex K_0 into K_1 we write $K_0 \searrow K_1$ and say that K_0 collapses to K_1 . If K_1 is one-dimensional, we say that K_0 is erasable. If K_1 is merely a point we call K_0 collapsible.

Finally, we write $K_0 \nearrow K_1$ to indicate a sequence of expansions and say that K_0 expands to K_1 .

It follows that an expansion \nearrow_σ^τ can only be performed in a simplicial complex K if all codimension 1 faces of τ except for σ are already in K . Hence, let τ be an m -face of a simplicial complex K , and let σ be one of its $(m - 1)$ -faces. An (*m -dimensional*) *horn* $\mathbf{H}(\tau, \sigma)$ associated to the pair (τ, σ) is the simplicial complex generated by the $(m - 1)$ -faces of τ apart from σ .

13:4 Parametrized Complexity of Expansion Height

All m -expansions and m -collapses with $m > 1$ leave the vertex set unchanged.

► **Definition 2** (Simple-homotopy equivalence, simple-homotopy graph). *Two simplicial complexes K and L are said to be simple homotopy equivalent or of coinciding simple-homotopy type, written $K \frown L$, if there exists a sequence \mathcal{S} of elementary moves turning one into the other. In this case, we write $\mathcal{S} : K \frown L$.*

The dimension of a simple-homotopy equivalence is the maximum of the dimensions of K , L and of any elementary expansion or collapse in the sequence.

The graph whose nodes are simplicial complexes, and two nodes are adjacent if their corresponding complexes are related by an elementary collapse is called simple-homotopy graph. Naturally, its connected components are in one-to-one correspondence with simple-homotopy types.

Two simplicial complexes of the same simple-homotopy type are homotopy equivalent, but the converse is not true, see, for instance, [18]. For simple-homotopy equivalent simplicial complexes we know the following.

► **Theorem 3** (Wall [17], Matveev [13, Theorem 1.3.5]). *Let K and L be two simplicial complexes of the same simple-homotopy type and of dimension at most $m > 2$. Then there exists a simple-homotopy equivalence of dimension at most $m + 1$, taking one to the other.*

For the case $m = 2$, Theorem 3 is still open and known as the (topological or geometric) Andrews–Curtis conjecture [2, 13, 15]. On the other hand, it is known that any contractible 2-complex is also simple-homotopy equivalent to a point [19]. Hence, any pair of contractible 2-complexes can be connected by a simple-homotopy equivalence of dimension at most four – but determining whether we can always decide if such a simple-homotopy equivalence exists is an open question [4], equivalent to the triviality problem for balanced group presentations [12].

2.3 Parametrized complexity

Parametrized complexity, as introduced by Downey and Fellows in [7], is a refinement of classical complexity theory. The theory revolves around the general idea of developing complexity bounds for instances of a problem not just based on their size, but also involving an additional *parameter*, which might be significantly smaller than the size. Specifically, we have the following definition.

► **Definition 4** (Parameter, parametrized problem). *Let Σ be a finite alphabet.*

1. *A parameter of Σ^* , the set of strings over Σ , is a function $\rho : \Sigma^* \rightarrow \mathbb{N}$, attaching to every input $w \in \Sigma^*$ a natural number $\rho(w)$.*
2. *A parametrized problem over Σ is a pair (P, ρ) consisting of a set $P \subseteq \Sigma^*$ and a parametrization $\rho : \Sigma^* \rightarrow \mathbb{N}$.*

In this article we consider the complexity class $\mathbf{W[P]}$ for parametrized problems, following the definition by Flum and Grohe [8].

► **Definition 5** (Complexity Class $\mathbf{W[P]}$). *Let Σ be an alphabet and $\rho : \Sigma^* \rightarrow \mathbb{N}$ a parametrization. A nondeterministic Turing machine \mathbb{M} with input alphabet Σ is called ρ -restricted if there are computable functions $f, h : \mathbb{N} \rightarrow \mathbb{N}$ and a polynomial p (with coefficients in the set of natural numbers) such that on every run with input $x \in \Sigma^*$ the machine \mathbb{M} performs at most $f(k) \cdot p(|x|)$ steps, at most $h(k) \cdot \log |x|$ of them being nondeterministic, where $k := \rho(x)$. $\mathbf{W[P]}$ is the class of all parametrized problems (P, ρ) that can be decided by a ρ -restricted nondeterministic Turing machine.*

3 Problems

In this article we consider the following parametrized problems.

► **Problem 1** (ERASIBILITY EXPANSION HEIGHT).

INSTANCE: A 2-dimensional simplicial complex K and a natural number p .

PARAMETER: p .

QUESTION: Is there a path in the simple-homotopy graph connecting K to a 1-complex using at most p expansions?

► **Problem 2** (ORDERED ERASIBILITY EXPANSION HEIGHT).

INSTANCE: A 2-dimensional simplicial complex K and a natural number p .

PARAMETER: p .

QUESTION: Is there a path in the simple-homotopy graph connecting K to a 1-complex using first at most p expansions, followed by a sequence of only collapses?

In Section 5, we establish $\mathbf{W[P]}$ -completeness for ERASIBILITY EXPANSION HEIGHT and ORDERED ERASIBILITY EXPANSION HEIGHT.

The hardness proof works via a parametrized reduction using the AXIOM SET problem, which is a classical \mathbf{NP} -complete problem [9, p. 263] that is well-known to be $\mathbf{W[P]}$ -complete with respect to the appropriate parameter [7, p. 473].

► **Problem 3** (AXIOM SET).

INSTANCE: A finite set S of *sentences*, an *implication relation* R consisting of pairs (U, s) where $U \subseteq S$ and $s \in S$, and a positive integer $p \leq |S|$.

PARAMETER: p .

QUESTION: Is there a set $S_0 \subseteq S$, called an *axiom set*, with $|S_0| \leq p$ and a positive integer n such that if we recursively define

$$S_i := S_{i-1} \cup \{s \in S \mid \exists U \subseteq S_{i-1} : (U, s) \in R\}$$

for $1 \leq i \leq n$, then $S_n = S$?

► **Remark 6.** Note that every instance of AXIOM SET can be reduced in polynomial time to an instance for which every sentence must occur in at least one implication relation: First iteratively remove all sentences from the instance which do not feature in at least one implication relation. Then, for each of them, reduce p by one (note that each of them must necessarily be an axiom). It follows that solving the reduced instance is equivalent to solving the original instance.

Similarly, note that if there exists an implication $(U, s) \in R$, $s \in U$, we can simply omit it and, if this deletes s from the instance altogether, decrease p by one.

In Section 6, we show that the following variants of the expansion height problem are \mathbf{NP} -complete.

► **Problem 4** (ERASIBILITY 3-EXPANSION HEIGHT).

INSTANCE: A finite 2-dimensional simplicial complex K and a natural number p .

QUESTION: Is there a path in the simple-homotopy graph connecting K to a 1-complex using at most p expansions, all of which are 3-expansions?

► **Problem 5** (ORDERED ERASIBILITY 3-EXPANSION HEIGHT).

INSTANCE: A finite 2-dimensional simplicial complex K and a natural number p .

QUESTION: Is there a path in the simple-homotopy graph connecting K to a 1-complex using first at most p expansions, all of which are 3-expansions, followed by a sequence of only collapses?

4 Contractibility and collapsibility for 2-complexes

The main gadget used in the proof of our main result, Theorem 8, is based on the simplest 2-dimensional contractible complex which is not collapsible to a point – the *dunce hat*. Hence, before we describe our main gadget in detail, we start this section by briefly discussing minimal triangulations of the dunce hat, and a variant that is collapsible through a unique free edge, the *modified dunce hat*.

4.1 The dunce hat

In the category of CW complexes, the dunce hat can be obtained by identifying two boundary edges of a triangle to build a cone and then gluing the third edge along the seam of the first gluing. The resulting complex does not have a collapsible triangulation. On the other hand, the dunce hat is known to be contractible [21].

The smallest simplicial complexes realizing this construction have 8 vertices, 24 edges and 17 triangles. There are seven such minimal triangulations of the dunce hat [16]. One such triangulation, denoted by **D**, is shown in Figure 1. The dunce hat **D** has two horns, namely $\mathbf{H}(\{2, 7, 8\}, \{1, 2, 7, 8\})$ and $\mathbf{H}(\{3, 5, 6\}, \{1, 3, 5, 6\})$, and hence admits two 3-expansions, namely $\nearrow_{\{2,7,8\}}^{\{1,2,7,8\}}$ and $\nearrow_{\{3,5,6\}}^{\{1,3,5,6\}}$ respectively. They are shown by the shaded areas in Figure 1.

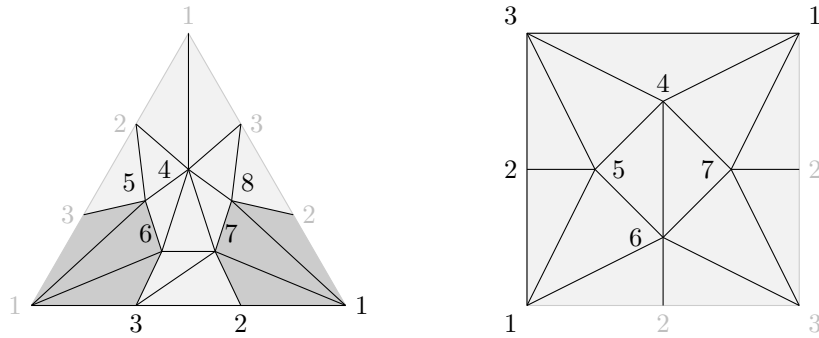


Figure 1 Left: The 8-vertex triangulation **D** of the dunce hat. The two expansions turning it collapsible are highlighted. Right: The 7-vertex triangulation **P** of the modified dunce hat.

Note that after any of these two expansions we obtain a collapsible complex: After the expansion $\nearrow_{\{2,7,8\}}^{\{1,2,7,8\}}$ and the collapses $\searrow_{\{1,7,8\}}^{\{1,2,7,8\}}$, $\searrow_{\{1,7\}}^{\{1,2,7\}}$, and $\searrow_{\{1,8\}}^{\{1,2,8\}}$, the edge $\{1, 2\}$ becomes free and thus **D** becomes collapsible. Similarly, starting with $\nearrow_{\{3,5,6\}}^{\{1,3,5,6\}}$, one may perform the collapse $\searrow_{\{1,5,6\}}^{\{1,3,5,6\}}$ and proceed in an analogous way. In particular, this shows that the dunce hat has the simple-homotopy type of a point, and in fact can be made collapsible by using a single expansion.

4.2 The modified dunce hat

Rather than working with the dunce hat directly, we base the construction of our gadget for the proof of Theorem 8 on the *modified dunce hat* [10]. More precisely, we “insert” a free edge into the dunce hat. For instance, Figure 1 depicts a triangulation of the modified dunce hat, which we denote by **P**, with $\{1, 3\}$ as the unique free edge. This particular triangulation of the modified dunce hat uses only 7 vertices, 19 edges, and 13 triangles. The modified dunce hat has previously been used as a gadget to show hardness of approximation for Morse matchings [3].

If we assume that \mathbf{P} is part of a larger complex K , in which edge $\{1, 3\}$ is glued to triangles not lying in \mathbf{P} , then $\{1, 3\}$ is not free. In this case, the triangles of \mathbf{P} can be collapsed away in essentially two distinct ways. Either, at some point in a simple-homotopy on K , the edge $\{1, 3\}$ becomes free and thus the triangles of \mathbf{P} collapse, or the triangles of \mathbf{P} become collapsible by performing one of two possible 3-expansions on \mathbf{P} . Looking at the latter case in more detail, we have the following sequences of expansions and collapses:

$$\nearrow_{\{2,5,6\}}^{\{1,2,5,6\}}, \searrow_{\{1,5,6\}}^{\{1,2,5,6\}}, \searrow_{\{1,5\}}^{\{1,2,5\}}, \searrow_{\{1,6\}}^{\{1,2,6\}}$$

and

$$\nearrow_{\{2,6,7\}}^{\{2,3,6,7\}}, \searrow_{\{3,6,7\}}^{\{2,3,6,7\}}, \searrow_{\{3,6\}}^{\{2,3,6\}}, \searrow_{\{3,7\}}^{\{2,3,7\}}.$$

In the first case, the edge $\{1, 2\}$ is freed, in the second case, the edge $\{2, 3\}$ is freed. Both sequences can be extended to a collapsing sequence of the entire complex \mathbf{P} .

4.3 The main gadget

Our gadget for the proof of Theorem 8 is a subdivided version of the modified dunce hat \mathbf{P} from Section 4.2. More precisely, it is determined by two positive integers m and ℓ , denoted by $\mathbf{P}_{m,\ell}$, and can be constructed from the complex \mathbf{P} in essentially two steps.

1. Subdivide the edge $\{1, 3\}$ of \mathbf{P} ($m - 1$) times, thereby introducing vertices x_1, \dots, x_{m-1} . Relabel $1 \rightarrow x_0$ and $3 \rightarrow x_m$ to obtain m free edges $f_i = \{x_{i-1}, x_i\}$, $1 \leq i \leq m$.
2. Remove the edge $\{4, 6\}$ and place ℓ vertex-disjoint copies of the disk

$$\{\{c_j, a_j, y_j\}, \{c_j, y_j, z_j\}, \{c_j, z_j, b_j\}, \{d_j, a_j, y_j\}, \{d_j, y_j, z_j\}, \{d_j, z_j, b_j\}\},$$

$1 \leq j \leq \ell$, inside the 4-gon in the center of \mathbf{P} bounded by 4, 5, 6, and 7. Triangulate the remaining space in the interior of the 4-gon. This creates edges $e_j = \{y_j, z_j\}$, $1 \leq j \leq \ell$, with pairwise vertex disjoint stars disjoint to 4, 5, 6, and 7 (now $\{4, 6\}$ reappears as a path from 4 to 6, and thus $\mathbf{P}_{m,\ell}$ is in fact a proper subdivision of \mathbf{P}). See Figure 2 for an illustration.

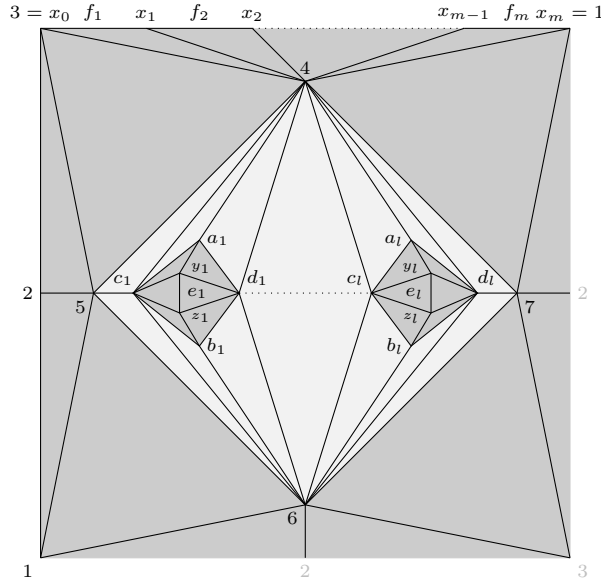
One key property of $\mathbf{P}_{m,\ell}$ is that we do not subdivide any faces of \mathbf{P} near to the two available 3-expansions. As a result, again, assuming that $\mathbf{P}_{m,\ell}$ is part of a larger complex K where all free edges of $\mathbf{P}_{m,\ell}$ are glued to other triangles of K outside of $\mathbf{P}_{m,\ell}$ and thus are not free, the triangles of $\mathbf{P}_{m,\ell}$ can be collapsed according to the following observation:

► Remark 7. Let K be a two-dimensional simplicial complex such that $\mathbf{P}_{m,\ell}$ is a subcomplex whose vertices do not span any other faces of K (i.e., $\mathbf{P}_{m,\ell}$ is an induced subcomplex of K), and $K \frown_{\searrow} L$ where L is a 1-complex. Then, at least one of the following three statements holds true at some point in $K \frown_{\searrow} L$, enabling us to eventually collapse away all the triangles of $\mathbf{P}_{m,\ell}$.

1. one of the edges $f_i \in \mathbf{P}_{m,\ell}$ becomes free;
2. one of two 3-expansions on $\mathbf{P}_{m,\ell}$: $\nearrow_{\{2,5,6\}}^{\{1,2,5,6\}}$ or $\nearrow_{\{2,6,7\}}^{\{2,3,6,7\}}$ is performed;
3. multiple expansions result in a complex in which all the triangles of $\mathbf{P}_{m,\ell}$ can be collapsed.

In other words, if one of the edges $f_i \in \mathbf{P}_{m,\ell}$ does not become free at some point in $K \frown_{\searrow} L$, then one is forced to use 3-expansions (either directly on $\mathbf{P}_{m,\ell}$, or after performing additional expansions) to collapse away the triangles of $\mathbf{P}_{m,\ell}$.

In Section 5.1 we use this gadget to reduce an instance $A = (S, R, p)$ of AXIOM SET to ERASIBILITY EXPANSION HEIGHT: Every sentence $s \in S$ is associated with one copy of $\mathbf{P}_{m,\ell}$, the edges f_i correspond to implications $(U, s) \in R$, and the edges e_j correspond to whenever $s \in U$ for some implication $(U, u) \in R$.



■ **Figure 2** The main gadget of the construction $\mathbf{P}_{m,\ell}$.

5 Parametrized complexity of Erasability Expansion Height

In this section, we first prove that ERASIBILITY EXPANSION HEIGHT and ORDERED ERASIBILITY EXPANSION HEIGHT are $\mathbf{W[P]}$ -hard by a reduction from AXIOM SET, a problem known to be $\mathbf{W[P]}$ -complete. We then show that the two problems are also contained in $\mathbf{W[P]}$.

5.1 $\mathbf{W[P]}$ -hardness of expansion height problems

► **Theorem 8.** ERASIBILITY EXPANSION HEIGHT and ORDERED ERASIBILITY EXPANSION HEIGHT are $\mathbf{W[P]}$ -hard problems.

The following lemma is used to assemble the gadgets in our reduction into a simplicial complex K .

► **Lemma 9** (Munkres, [14, Lemma 3.2]). *Let C be a finite set, let K be a simplicial complex with set of vertices V , and let $f : V \rightarrow C$ be a surjective map associating to each vertex of K a color from C . The coloring f extends to a simplicial map $g : K \rightarrow K_f$ where K_f has vertex set C and is obtained from K by identifying vertices with equal color.*

If for all pairs $v, w \in V$, $f(v) = f(w)$ implies that their stars $\text{star}_K(v)$ and $\text{star}_K(w)$ are vertex disjoint, then, for all faces $\tau, \sigma \in K$ we have that

- τ and $g(\tau)$ have the same dimension, and
- $g(\tau) = g(\sigma)$ implies that either $\tau = \sigma$ or τ and σ are vertex disjoint in K .

Lemma 9 provides a way of gluing faces of a simplicial complex by a simplicial quotient map obtained from vertex identifications, and tells us when this gluing does not create unwanted identifications.

Proof of Theorem 8. We want to reduce AXIOM SET to ERASIBILITY EXPANSION HEIGHT.

Fix an instance $A = (S, R, p)$ of AXIOM SET such that every sentence $s \in S$ is subject to at least one implication $(U, s) \in R$ and such that $(U, s) \in R$ implies $s \notin U$. By Remark 6, this is not a restriction, since every instance of AXIOM SET can be reduced to such an

instance in polynomial time. For every sentence $s \in S$, take a copy \mathbf{P}_s of the gadget $\mathbf{P}_{m,\ell}$ to model s , where $\ell \geq 0$ is the number of implications $(U, u) \in R$ with $s \in U$ and $m \geq 1$ is the number of implications $(U, s) \in R$. Thus, for all values $\ell \geq 0$ and $m \geq 1$ the gadget $\mathbf{P}_{m,\ell}$ is a simplicial complex without any unintended identifications. Denote the free edges of \mathbf{P}_s by $f_i^s = \{x_{i-1}^s, x_i^s\}$, $1 \leq i \leq m$, and its edges of type e_j by $e_j^s = \{y_j^s, z_j^s\}$, $1 \leq j \leq \ell$.

For a fixed $s \in S$, endow the set of implications $(U, s) \in R$ of s with an arbitrary order $(U_1, s), \dots, (U_m, s)$. Similarly, for a fixed $u \in S$, order the set of implications in R containing u arbitrarily as $(U^1, s^1), \dots, (U^\ell, s^\ell)$. Now for every $(U_i, s) \in R$ and every $u \in U_i = U^j$ (i.e., $s^j = s$), glue the edge $f_i^s = \{x_{i-1}^s, x_i^s\}$ of the gadget \mathbf{P}_s to the edge $e_j^u = \{y_j^u, z_j^u\}$ of \mathbf{P}_u by identifying x_{i-1}^s with y_j^u and x_i^s with z_j^u .

Performing these identifications for all implications in R yields a complex, which we denote by K . Note that, fixing $s \in S$, and $0 \leq i \leq m$, the only vertices to which x_i^s can possibly be identified to in K are y_j^u, z_j^u ($1 \leq j \leq \ell$).

More precisely, using the orderings from above for vertex x_i^s , let (U_i, s) and (U_{i+1}, s) be the i -th and $(i+1)$ -st implication of s in R (if $i \in \{0, m\}$ there is only one implication to consider) and denote their sentences by u_1, \dots, u_r and u^1, \dots, u^t , where $r = |U_i|$ and $t = |U_{i+1}|$. Moreover, let U_i (resp. U_{i+1}) be the j_f -th (resp. j_g -th) implication where the sentence u_f (resp. u^g) occurs, for $1 \leq f \leq r$ (resp. $1 \leq g \leq t$). Then x_i^s is identified with $z_{j_f}^{u_f}$ ($1 \leq f \leq r$) and $y_{j_g}^{u^g}$ ($1 \leq g \leq t$).

Now since every fixed edge of type e_j^u is only identified with one edge of type f_i^s , those vertices are not identified with any other vertices of K . Since, by construction, the set of the vertex stars of $z_{j_f}^{u_f}$ ($1 \leq f \leq r$), $y_{j_g}^{u^g}$ ($1 \leq g \leq t$), and x_i^s are pairwise vertex disjoint, we can apply Lemma 9 to ensure that no unwanted identifications occur in building up K . In particular, every gadget \mathbf{P}_s is a subcomplex of K via the canonical isomorphism given by the gluing map.

We now show that the following three statements are equivalent for our complex K :

- (a) there exists a simple-homotopy equivalence turning K into a 1-dimensional complex using first at most p expansions, followed by a sequence of only collapses,
- (b) there exists a simple-homotopy equivalence $K \searrow_\triangleleft L$ turning K into a 1-dimensional complex L using at most p expansions, and
- (c) there exists an axiom set $S_0 \subset S$ for $A = (S, R, p)$ using at most p elements.

We trivially have that (a) \implies (b).

In order to show that (c) \implies (a), assume that there exists an axiom set $S_0 \subset S$ of size p , and perform one 3-expansion on each gadget \mathbf{P}_u with $u \in S_0$. As described in Section 4.3, these expansions admit all triangles of these gadgets to collapse. This, in turn, frees all edges f_i^s where $s \in S$ has an implication $(U, s) \in R$ with $U \subset S_0$. Consequently, all triangles of such gadgets \mathbf{P}_s can be collapsed. Since S_0 is an axiom set, repeating this process eventually collapses away all tetrahedra and triangles, leaving a 1-complex.

In order to show that (b) \implies (c), we start with a few definitions. For a complex K' with $K \searrow_\triangleleft K'$, we say that a gadget $\mathbf{P}_s \subseteq K$ is *touched* with respect to a simple-homotopy sequence $\mathcal{S} : K \searrow_\triangleleft K'$ if at some point in the simple-homotopy sequence one of the triangles of \mathbf{P}_s is removed. Otherwise \mathbf{P}_s is said to be *untouched*. Note that even if all triangles of \mathbf{P}_s are present in K' , \mathbf{P}_s might still be touched. Being touched or untouched is a property of the sequence $\mathcal{S} : K \searrow_\triangleleft K'$, not of the complex K' .

We build the axiom set $S_0 \subset S$ for $A = (S, R, p)$ in the following way: A sentence $s \in S$ is in S_0 if and only if a triangle in \mathbf{P}_s is removed by a 3-collapse of the given simple-homotopy sequence $\mathcal{S} : K \searrow_\triangleleft L$. We first inductively prove a claim about

$$S^k = \{s \in S \mid \mathbf{P}_s \text{ is touched by a 3-collapse in the first } k \text{ moves of } K \searrow_\triangleleft L\}.$$

13:10 Parametrized Complexity of Expansion Height

We first inductively prove a claim about

$$S^k = \{s \in S_0 \mid \mathbf{P}_s \text{ is by a 3-collapse in the first } k \text{ moves of } K \frown L\}.$$

▷ **Claim 10.** For $s \in S$, if the gadget \mathbf{P}_s is touched by the first k elementary moves in $\mathcal{S} : K \frown L$, then s is implied by sentences in S^k .

Proof. First note that all gadgets are untouched in K and $S^0 = \emptyset$.

By induction hypothesis, if a gadget \mathbf{P}_s is touched by one of the first $k - 1$ moves in $\mathcal{S} : K \frown L$, then s is implied by sentences in S^{k-1} .

The induction claim is trivially true if \mathbf{P}_s is touched in the first $k - 1$ moves, or if \mathbf{P}_s is touched by a 3-collapse in the k -th move ($s \in S^k \setminus S^{k-1}$), causing the sentence s to be included in S_0 .

So, suppose that this is not the case. That is, suppose that \mathbf{P}_s is untouched in the length $k - 1$ prefix $\mathcal{S}' : K \frown K'$ of $\mathcal{S} : K \frown L$ and touched by a 2-collapse in the k -th move. This implies that $S^k = S^{k-1}$ and that \mathbf{P}_s is a subcomplex of K' and one of the edges f_i^s must be free in K' . Now let $\mathbf{P}_{u_1}, \mathbf{P}_{u_2}, \dots, \mathbf{P}_{u_q}$ be the set of other gadgets containing triangles glued to f_i^s in the original complex K (that is, there is an implication $(\{u_1, u_2, \dots, u_q\}, s) \in R$). Since none of these triangles are present in K' , all gadgets $\mathbf{P}_{u_1}, \mathbf{P}_{u_2}, \dots, \mathbf{P}_{u_q}$ must be touched in $\mathcal{S}' : K \frown K'$. Thus, either they were touched by a 3-collapse and their corresponding sentences are part of S^{k-1} , or they were touched by a 2-collapse and, by the induction hypothesis, their corresponding sentences are implied by sentences in S^k . It follows that s is implied by sentences in $S^k = S^{k-1}$, proving the claim. ◁

By assumption, K is simple homotopy equivalent to a 1-complex L . That is, $\mathcal{S}' : K \frown L$ eventually removes all triangles from K . Hence, every sentence $s \in S$ is touched as a result of a 2-collapse or a 3-collapse. Let m be the number of elementary moves needed to reach L starting from K . Then, by Claim 10, $S^m = S_0$ is the desired axiom set. Also, since a sentence s is included in S_0 only if a triangle belonging to gadget \mathbf{P}_s is removed as part of a 3-collapse, and since a triangle belonging to gadget \mathbf{P}_s does not belong to any other gadget \mathbf{P}_u for $u \neq s$, S_0 cannot contain more elements than the number of 3-collapses (and hence 3-expansions).

Finally, we infer $\mathbf{W}[\mathbf{P}]$ -hardness of ERASIBILITY EXPANSION HEIGHT and ORDERED ERASIBILITY EXPANSION HEIGHT from the above equivalence and the $\mathbf{W}[\mathbf{P}]$ -hardness of AXIOM SET [7, p. 473]. ◀

5.2 $\mathbf{W}[\mathbf{P}]$ -membership of Erasibility Expansion Height

We now show that ORDERED ERASIBILITY EXPANSION HEIGHT and ERASIBILITY EXPANSION HEIGHT are in $\mathbf{W}[\mathbf{P}]$ by describing suitable nondeterministic algorithms for deciding both problems. We begin with a well-known fact about checking collapsibility of 2-complexes.

► **Proposition 11** (Tancer [17], Proposition 5). *Let K be a 2-complex that collapses to a 1-complex L and to another 2-complex M . Then M also collapses to a 1-complex.*

► **Remark 12.** The proposition above implies that we can collapse an input 2-complex K greedily until no more 2-collapses are possible, and if K collapses to a 1-complex L , the algorithm is guaranteed to terminate with a 1-complex as well.

► **Theorem 13.** ORDERED ERASIBILITY EXPANSION HEIGHT is in $\mathbf{W}[\mathbf{P}]$.

Proof. Let K be a simplicial complex with n simplices. First, note that if there exists a simple homotopy sequence \mathcal{S} taking K to a 1-complex with p expansions that all come at the beginning of the sequence, then there also exists a simple homotopy sequence $\mathcal{S}_{\mathbb{M}}$ taking K to a 1-complex where p expansions are followed by collapses such that, for each d , all collapses of dimension $d + 1$ are executed before collapses of dimension d . This follows from observing that, for any two d -collapses \searrow_{σ}^{τ} and $\searrow_{\alpha}^{\beta}$, if the d -collapse \searrow_{σ}^{τ} is executed before the d -collapse $\searrow_{\alpha}^{\beta}$ in \mathcal{S} , then the same can be carried out in $\mathcal{S}_{\mathbb{M}}$. Also, in any simple homotopy sequence \mathcal{S} that takes K to a 1-complex, for every $d > 2$, the number of d -expansions equals the number of d -collapses in \mathcal{S} . This follows from a simple inductive argument starting with highest dimensional moves.

Denoting the total number of d -collapses, $d > 2$, in \mathcal{S} by $q \leq p$, it follows that, if there exists a simple homotopy sequence \mathcal{S} with p expansions that come at the beginning, then there exists a simple homotopy sequence $\mathcal{S}_{\mathbb{M}}$ with p expansions in the beginning followed by q collapses that gives rise to a 2-complex K' with $O(n^3)$ faces. The faces can be as many as $O(n^3)$ since \mathbb{M} does not guess any 2-collapses. Furthermore, if \mathcal{K} is erasable through the simple homotopy sequence \mathcal{S} , then K' is also erasable, once again, because the 2-collapses of \mathcal{S} can be carried out in the same order in $\mathcal{S}_{\mathbb{M}}$. Hence, the non-deterministic Turing machine \mathbb{M} can now be described as follows:

1. Guess p expansions and q collapses non-deterministically to obtain a complex K' .
2. Deterministically check if K' is erasable.

By Remark 12, erasability of K' can be deterministically checked in time polynomial in n .

Since any simplex in the desired simple homotopy sequence has at most $n + p$ vertices, the number of bits required to encode a single vertex is $O(\log(n + p))$. Also, because the dimension of the faces involved in expansions and collapses is certainly in $O(p)$, and since an expansion or a collapse can be fully described by a pair of simplices, the number of bits required to encode an expansion or a collapse is $O(p \log(n + p))$. Hence, in order to guess $p + q$ moves, it suffices for \mathbb{M} to guess $O(p^2 \cdot \log(n + p))$ bits in total since $q \leq p$. Now, assuming $n, p \geq 2$, we have

$$p^2 \log(n + p) \leq p^2 \log(np) = p^2 \log(n) + p^2 \log(p) \leq p^2(1 + \log(p)) \log(n).$$

Hence, for sufficiently large n and p , the number of bits guessed by \mathbb{M} is bounded by a function of the form $f(p) \log n$. Thus, \mathbb{M} is a p -restricted Turing machine, and ORDERED ERASIBILITY EXPANSION HEIGHT is in $\mathbf{W}[P]$. ◀

► **Theorem 14.** ERASIBILITY EXPANSION HEIGHT is in $\mathbf{W}[P]$.

Proof. Assume that there exists a simple homotopy that takes K to a 1-complex using no more than p expansions. The Turing machine \mathbb{M} needs to generate one such sequence. Below, we show that, in order to achieve this, \mathbb{M} does not have to guess an entire simple homotopy sequence, but only a subsequence, and the remaining part of the sequence can be found deterministically by \mathbb{M} .

Given a 2-dimensional complex K with n faces, \mathbb{M} first nondeterministically guesses p expansions and p collapses, and the order in which they are to be executed. These moves are referred to in the following as *prescribed moves*. While these moves are meant to appear in a specified order, they need not appear consecutively. The moves that are not prescribed are computed deterministically by \mathbb{M} . A simple homotopy sequence of K , that takes K to a 1-complex, in which all the prescribed moves occur as a subsequence, is called a sequence *compatible* with the prescribed moves. By assumption, there exists a set of prescribed moves for which a compatible sequence exists.

13:12 Parametrized Complexity of Expansion Height

In order to give a description of \mathbb{M} , we introduce some additional terminology. Let \mathcal{S}_q^j be an intermediate simple homotopy sequence computed by \mathbb{M} , such that the first j prescribed moves guessed by \mathbb{M} form a subsequence of \mathcal{S}_q^j , q is the total number of moves in \mathcal{S}_q^j , and \mathcal{S}_q^j is a prefix of a set of sequences \mathcal{S} compatible with the prescribed set of moves. Let $K \frown_{\searrow} K_q^j$ be the complex obtained by executing the moves in \mathcal{S}_q^j . Then, a collapse \searrow_{σ}^{τ} in K_q^j is *valid* for this prefix if appending the collapse still leaves a compatible prefix. That is, \mathcal{S}_q^j appended with the collapse \searrow_{σ}^{τ} (giving \mathcal{S}_{q+1}^j) continues to be a prefix of at least one compatible sequence $\mathcal{S} \in \mathcal{S}$. A collapse that is not valid is said to be *forbidden*.

Note that labelling vertices of a complex C by natural numbers determines a lexicographic order on the simplices of C . The lexicographic order $<_C$ on simplices of C can be extended to a lexicographic order \prec on collapses as follows: If $(\searrow_{\sigma}^{\tau}), (\searrow_{\alpha}^{\beta})$ are two collapses in C , then $(\searrow_{\sigma}^{\tau}) \prec (\searrow_{\alpha}^{\beta})$ if $\sigma <_C \alpha$.

The Turing machine \mathbb{M} for deciding ERASIBILITY EXPANSION HEIGHT can be described as follows:

1. Guess $2p$ prescribed moves non-deterministically.
2. Execute 2-collapses in lexicographic order until no more 2-collapses are valid.
3. Repeat until all prescribed moves have been executed:
 - a. Execute the next prescribed move.
 - b. Execute 2-collapses in lexicographic order until no more collapses are valid.

Let \mathcal{S} be a sequence compatible with an ordered set of prescribed moves X (of cardinality $2p$), and let $\mathcal{S}_{\mathbb{M}}$ be a simple homotopy sequence computed by \mathbb{M} as above such that X is a subsequence of $\mathcal{S}_{\mathbb{M}}$. Now, let σ be a free edge associated with a 2-collapse \searrow_{σ}^{τ} in \mathcal{S}_q^j for some j and q , where \mathcal{S}_q^j is a subsequence of $\mathcal{S}_{\mathbb{M}}$. If there exist future prescribed moves including cofaces of σ , then the next prescribed move including cofaces of σ that is not an expansion involving σ is denoted by m_1 . Similarly, if there exist future prescribed moves including cofaces of τ , then the next prescribed move including cofaces of τ that is not an expansion involving τ is denoted by m_2 . Note that m_2 cannot come before m_1 but we may have $m_1 = m_2$. Then, the 2-collapse \searrow_{σ}^{τ} is forbidden if and only if m_1 exists and is not preceded by a future prescribed expansion involving σ or m_2 exists, and is not preceded by a future prescribed expansion involving τ . It follows that, for each free edge, checking if a collapse is forbidden (or valid) can be done deterministically in time polynomial in p . To see this note that the most expensive atomic operation is to check if a simplex (of dimension 1 or 2) is a face of a simplex that is at most p dimensional, and the number of prescribed moves is at most $2p$. Altogether, the set of valid collapses can be computed in time polynomial in n and p , which can also be lexicographically ordered in polynomial time.

Finally, let K' denote the complex obtained from K by the sequence $\mathcal{S}_{\mathbb{M}}$. Then, the following claim establishes the effectiveness of the greedy strategy employed by \mathbb{M} .

▷ **Claim 15.** If there exists a simple homotopy sequence with at most p expansions that takes K to a 1-complex, then there exists an execution branch of the Turing machine that terminates successfully, i.e., the complex K' obtained by \mathbb{M} is a 1-complex.

Proof. Let \mathcal{S} be a simple homotopy sequence with p expansions that takes K to a 1-complex. Let X_e be the ordered set of expansions in \mathcal{S} . Thus, $|X_e| = p$. Moreover, let X_e^+ (X_e^-) denote the d -expansions (d -collapses) in \mathcal{S} with $d > 2$. As in Theorem 13, by a simple inductive argument starting from the highest dimension it can be shown that $|X_c^+| = |X_e^+|$. To the p expansions of \mathcal{S} , we associate a set X_c of collapses of \mathcal{S} as follows: If $|X_c^+| < p$, then let X_c^- be an arbitrary set of d -collapses in \mathcal{S} with $d \leq 2$, and $|X_c^-| = p - |X_c^+|$. Now, let $X_c = X_c^+ \cup X_c^-$, so that $|X_c| = p$. Finally, let the ordered set X of prescribed moves be the set containing all elements of $X_c \cup X_e$ seen as a subsequence of \mathcal{S} .

We assume that the non-deterministic Turing machine \mathbb{M} correctly guesses the specified sequence of prescribed moves X . It now suffices to show the following claim about the sequence $\mathcal{S}_{\mathbb{M}}$. \triangleleft

\triangleright **Claim 16.** $\mathcal{S}_{\mathbb{M}}$ is compatible with the prescribed moves X , and $\mathcal{S}_{\mathbb{M}}$ takes K to a 1-complex if \mathcal{S} takes K to a 1-complex.

Proof. Let $K \searrow_{\mathcal{S}} K^j$ denote the complex obtained from \mathcal{S} after executing the j -th prescribed move in \mathcal{S} . We show that there exists a complex $K_{\mathbb{M}}^j$ obtained from $\mathcal{S}_{\mathbb{M}}$ after executing the j -th prescribed move in $\mathcal{S}_{\mathbb{M}}$. Also, let $T^j (T_{\mathbb{M}}^j)$ denote the set of 2-simplices of $K^j (K_{\mathbb{M}}^j)$.

First observe that, $K_{\mathbb{M}}^0 = K$ exists and that $T_{\mathbb{M}}^0 = T^0$. We now show that the following claim is inductively true: $T_{\mathbb{M}}^j \subset T^j$ for all $j \in [1, 2p]$. Suppose we make the induction hypothesis that $T_{\mathbb{M}}^{j-1} \subset T^{j-1}$ for some $j \in [1, 2p]$. Then, the set of forbidden collapses for \mathcal{S} and $\mathcal{S}_{\mathbb{M}}$ are the same until the j -th move in X can be reached. Let τ_1 be the first 2-face of K^{j-1} that is removed as part of a 2-collapse after $j-1$ prescribed moves have been executed in \mathcal{S} . Without loss of generality, assume that the 2-collapse that removes τ_1 is non-prescribed. Then, there exists an edge $\sigma_1 \subset \tau_1$ such that τ_1 is the unique coface of σ_1 in K^{j-1} . By induction hypothesis, since $T_{\mathbb{M}}^{j-1} \subset T^{j-1}$ the same is also true for $K_{\mathbb{M}}^{j-1}$. Since \mathbb{M} greedily removes every valid collapse it can (in lexicographic order), at some appropriate lexicographic index, τ_1 is also removed from $K_{\mathbb{M}}^{j-1}$ (possibly along with σ_1). Now, let $\tau_1, \tau_2, \dots, \tau_{q-1}$ be the first $q-1$ 2-faces removed from K^{j-1} (as part of non-prescribed collapses). Assume that $\tau_1, \tau_2, \dots, \tau_{q-1}$ have also been removed from $K_{\mathbb{M}}^{j-1}$. By the same reasoning as before, if τ_q is the q -th face to be removed from K^{j-1} (as part of non-prescribed collapses), then τ_q may also be removed from $\mathcal{S}_{\mathbb{M}}$ as part of a valid collapse. Hence, by induction, $T_{\mathbb{M}}^j \subset T^j$ for all $j \in [1, 2p]$.

Finally, since by assumption, K^{2p} collapses to a 1-complex, by applying arguments analogous to the induction above, the same is true for $K_{\mathbb{M}}^{2p}$ since $T_{\mathbb{M}}^{2p} \subset T^{2p}$. \triangleleft

Since given a 2-complex with n faces, \mathbb{M} non-deterministically guesses $2p$ moves, as in Theorem 13, the number of bits guessed by \mathbb{M} is bounded by $f(p) \log(n)$, where $f(p) = O(p^2(1 + \log(p)))$. Hence, \mathbb{M} is a p -restricted Turing machine, and ERASIBILITY EXPANSION HEIGHT is in $\mathbf{W[P]}$. \blacktriangleleft

6 NP-completeness of Erasibility 3-Expansion Height

Note that the parametrized reduction from AXIOM SET to ERASIBILITY EXPANSION HEIGHT (and ORDERED ERASIBILITY EXPANSION HEIGHT) is also a polynomial-time reduction (or Karp reduction) from AXIOM SET to ERASIBILITY 3-EXPANSION HEIGHT (and ORDERED ERASIBILITY 3-EXPANSION HEIGHT), since the complexity of reduction is independent of the parameter p and depends only on the size of the input complex. This observation leads us to the following result.

\blacktriangleright **Theorem 17.** *The decision problems ERASIBILITY 3-EXPANSION HEIGHT and ORDERED ERASIBILITY 3-EXPANSION HEIGHT are NP-hard.*

Proof. Since the AXIOM SET problem is known to be NP-hard [9], it follows that ERASIBILITY 3-EXPANSION HEIGHT and ORDERED ERASIBILITY 3-EXPANSION HEIGHT are also NP-hard. \blacktriangleleft

For the rest of the section, we assume that K is a 2-complex K with n faces and m vertices. The total number of simplices that one can encounter in any simple homotopy

13:14 Parametrized Complexity of Expansion Height

sequence of K using only 3-expansions is at most $M = O(m^4)$. (Note that the ground set of K is fixed since we do not allow 1-expansions). Hence, the total number of *elementary moves* that may be available at any given point in the sequence is bounded by $O(M)$. That is, p itself is bounded by $O(M)$.

► **Theorem 18.** ERASIBILITY 3-EXPANSION HEIGHT *is in NP*.

Proof. The non-deterministic algorithm \mathbb{M} for deciding ERASIBILITY 3-EXPANSION HEIGHT first guesses at each point in the simple homotopy sequence starting with K , one elementary move (out of at most $O(M)$ available moves), and constructs a new complex from the move. The total number of moves made by \mathbb{M} is bounded by $\binom{n+2p-1}{2}$. Finally, \mathbb{M} checks if the final complex is a 1-complex. ◀

► **Theorem 19.** ORDERED ERASIBILITY 3-EXPANSION HEIGHT *is in NP*.

Proof. The non-deterministic algorithm \mathbb{M} for deciding ORDERED ERASIBILITY 3-EXPANSION HEIGHT first guesses at most p 3-expansions followed by an equal number of 3-collapses, resulting in a 2-complex K' with n faces. From Remark 12, the erasability of K' can be deterministically checked in time polynomial in n , proving the claim. ◀

References

- 1 J. J. Andrews and M. L. Curtis. Free groups and handlebodies. *Proc. Amer. Math. Soc.*, 16:192–195, 1965. doi:10.2307/2033843.
- 2 Jonathan A. Barmak. *Algebraic topology of finite topological spaces and applications*, volume 2032 of *Lecture Notes in Mathematics*. Springer, Heidelberg, 2011. doi:10.1007/978-3-642-22003-6.
- 3 Ulrich Bauer and Abhishek Rathod. Hardness of approximation for Morse matching. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2663–2674. SIAM, Philadelphia, PA, 2019. doi:10.1137/1.9781611975482.165.
- 4 Gilbert Baumslag, Alexei G. Myasnikov, and Vladimir Shpilrain. Open problems in combinatorial group theory. Second edition. In *Combinatorial and geometric group theory (New York, 2000/Hoboken, NJ, 2001)*, volume 296 of *Contemp. Math.*, pages 1–38. Amer. Math. Soc., Providence, RI, 2002. doi:10.1090/conm/296/05067.
- 5 V. V. Borisov. Simple examples of groups with unsolvable word problem. *Mat. Zametki*, 6:521–532, 1969.
- 6 Benjamin A. Burton, Thomas Lewiner, João Paixão, and Jonathan Spreer. Parameterized complexity of discrete Morse theory. *ACM Trans. Math. Software*, 42(1):Art. 6, 24, 2016. doi:10.1145/2738034.
- 7 R. G. Downey and M. R. Fellows. *Parameterized complexity*. Monographs in Computer Science. Springer-Verlag, New York, 1999. doi:10.1007/978-1-4612-0515-9.
- 8 J. Flum and M. Grohe. *Parameterized complexity theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.
- 9 Michael R. Garey and David S. Johnson. *Computers and intractability*. W. H. Freeman and Co., San Francisco, Calif., 1979. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.
- 10 Masahiro Hachimori. *Combinatorics of constructible complexes*. PhD thesis, Tokyo University, 2000. URL: <https://pdfs.semanticscholar.org/81ee/09ed08ca6c7487e8e18639a7a05c110781c3.pdf>.
- 11 Allen Hatcher. *Algebraic topology*. Cambridge University Press, Cambridge, 2002.
- 12 Cynthia Hog-Angeloni and Wolfgang Metzler, editors. *Two-dimensional homotopy and combinatorial group theory*, volume 197 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, 1993. doi:10.1017/CB09780511629358.

- 13 Sergei Matveev. *Algorithmic topology and classification of 3-manifolds*, volume 9 of *Algorithms and Computation in Mathematics*. Springer, Berlin, second edition, 2007.
- 14 James R. Munkres. *Elements of algebraic topology*. Addison-Wesley Publishing Company, Menlo Park, CA, 1984.
- 15 Alexei D. Myasnikov, Alexei G. Myasnikov, and Vladimir Shpilrain. On the Andrews-Curtis equivalence. In *Combinatorial and geometric group theory (New York, 2000/Hoboken, NJ, 2001)*, volume 296 of *Contemp. Math.*, pages 183–198. Amer. Math. Soc., Providence, RI, 2002. doi:10.1090/conm/296/05074.
- 16 J. Paixão and J. Spreer. Random collapsibility and 3-sphere recognition, 2015. Preprint, 18 pages, 6 figures. arXiv:1509.07607.
- 17 Martin Tancer. Recognition of collapsible complexes is NP-complete. *Discrete Comput. Geom.*, 55(1):21–38, 2016. doi:10.1007/s00454-015-9747-1.
- 18 J. H. C. Whitehead. On incidence matrices, nuclei and homotopy types. *Ann. of Math. (2)*, 42:1197–1239, 1941. doi:10.2307/1970465.
- 19 J. H. C. Whitehead. Simple homotopy types. *Amer. J. Math.*, 72:1–57, 1950. doi:10.2307/2372133.
- 20 Perrin Wright. Group presentations and formal deformations. *Trans. Amer. Math. Soc.*, 208:161–169, 1975. doi:10.2307/1997282.
- 21 E. C. Zeeman. On the dunce hat. *Topology*, 2:341–358, 1964. doi:10.1016/0040-9383(63)90014-4.

UnLimited TRAnsfers for Multi-Modal Route Planning: An Efficient Solution

Moritz Baum

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
moritz.baum@kit.edu

Valentin Buchhold

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
buchhold@kit.edu

Jonas Sauer

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
jonas.sauer2@kit.edu

Dorothea Wagner

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
dorothea.wagner@kit.edu

Tobias Zündorf

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
zuendorf@kit.edu

Abstract

We study a multi-modal route planning scenario consisting of a public transit network and a transfer graph representing a secondary transportation mode (e.g., walking or taxis). The objective is to compute all journeys that are Pareto-optimal with respect to arrival time and the number of required transfers. While various existing algorithms can efficiently compute optimal journeys in either a pure public transit network or a pure transfer graph, combining the two increases running times significantly. As a result, even walking between stops is typically limited by a maximal duration or distance, or by requiring the transfer graph to be transitively closed. To overcome these shortcomings, we propose a novel preprocessing technique called ULTRA (UnLimited TRAnsfers): Given a complete transfer graph (without any limitations, representing an arbitrary non-schedule-based mode of transportation), we compute a small number of transfer shortcuts that are provably sufficient for computing all Pareto-optimal journeys. We demonstrate the practicality of our approach by showing that these transfer shortcuts can be integrated into a variety of state-of-the-art public transit algorithms, establishing the ULTRA-Query algorithm family. Our extensive experimental evaluation shows that ULTRA is able to improve these algorithms from limited to unlimited transfers without sacrificing query speed, yielding the fastest known algorithms for multi-modal routing. This is true not just for walking, but also for other transfer modes such as cycling or driving.

2012 ACM Subject Classification Theory of computation → Shortest paths; Mathematics of computing → Graph algorithms; Applied computing → Transportation

Keywords and phrases Algorithms, Optimization, Route Planning, Public Transportation

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.14

Related Version A full version of the paper is available at <https://arxiv.org/abs/1906.04832>.

Supplement Material Source code of ULTRA is available at <https://github.com/kit-algo/ULTRA>.

Funding This research was funded by the DFG under grant number WA 654123-2.



© Moritz Baum, Valentin Buchhold, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 14; pp. 14:1–14:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Research on efficient route planning algorithms has seen remarkable advances in the past decades. For many types of transportation networks, queries can be solved in a few milliseconds, even on a continental scale [3]. However, combining schedule-based (i.e., public transit) and non-schedule-based (e.g., walking, cycling, driving) transportation modes and solving the resulting multi-modal routing problem is still a challenge [26]. In this work, we consider a multi-modal problem that augments public transit with a *transfer graph*, which represents an arbitrary non-schedule-based mode of transportation that can be used for transferring between public transit stops. Given a source and target vertex in the transfer graph and a departure time, we want to compute all Pareto-optimal journeys regarding travel time and number of used public transit trips.

Related Work. Most algorithms for public transit routing either impose technical restrictions on the included transfers or have only been evaluated on networks featuring very sparse transfer graphs. Algorithms that were only evaluated for limited transfers include the graph-based techniques in [23] and [18], frequency-based search [5], Transfer Patterns [2] and its accelerated version, Scalable Transfer Patterns [4], Public Transit Labeling [11], and SUBITO [9]. A common restriction, employed by CSA [14, 15], RAPTOR [13], and their corresponding speedup techniques ACSA [25, 15] and HypRAPTOR [12], is to require that the transfer graph is transitively closed. This eliminates the need to search within the transfer graph, as every possible destination can be reached with a single edge. To ensure a reasonably sized transfer graph, transfers are typically limited by a maximal duration (e.g., 15 minutes of walking) or distance before the transitive closure is computed. As shown in [26], choosing a higher limit for the maximal transfer duration increases the size of the resulting transitively closed graph significantly. A limit of only 20 minutes on the maximal transfer duration already leads to a graph that is unsuitable for practical applications. A special case is Trip-Based Routing [27], which precomputes transfers between pairs of trips. This precomputation involves enumerating all possible transfers and then using a limited set of pruning rules to omit some, but not all unnecessary transfers. Trip-Based Routing was only evaluated for transitively closed transfer graphs and likely has prohibitively high preprocessing times on unrestricted transfer graphs.

Using a restricted transfer graph is often justified with the argument that long transfers are rarely useful. However, experiments performed in [26, 22] show that unrestricted walking often significantly reduces the travel time of optimal journeys. This effect is likely even stronger for faster transportation modes, such as bicycle or car. The only algorithms that can handle unrestricted transfer graphs so far are multi-modal techniques such as MCR [10] and UCCH [16]. These techniques work by interleaving a public transit routing algorithm with Dijkstra’s algorithm [17] on a contracted transfer network. Accordingly, they are fairly slow compared to pure public transit algorithms. Most recently HLRaptor and HLCSA [22] have been published. Here, RAPTOR/CSA are interleaved with Hub Labeling queries [1] instead of Dijkstra. While requiring more than an hour of preprocessing, this significantly improves query times. However, it is still not as efficient as pure public transit algorithms.

Our Contribution. Preliminary experiments [24] have shown that the impact of unrestricted transfers in Pareto-optimal journeys depends heavily on their position in the journey: *Initial transfers*, which connect the source to the first public transit vehicle, and *final transfers*, connecting the final vehicle to the target, are fairly common and often have a

large impact on the travel time. In contrast, *intermediate transfers* between public transit trips are only occasionally relevant for optimal journeys. This suggests that the number of unique paths in the transfer graph that occur as intermediate transfers of a Pareto-optimal journey is small. Using this insight, we propose a new preprocessing technique called ULTRA (UnLimited TRAnsfers), which computes a set of shortcut edges representing these paths. The preprocessing step is carefully engineered to ensure that the number of shortcuts remains small. Combined with efficient one-to-many searches for the initial and final transfers, these shortcuts are sufficient to answer all queries in the network correctly. ULTRA shortcuts can be used without adjustment by any algorithm that previously required a transitively closed transfer graph. Our experiments show that this enables unrestricted multi-modal queries with roughly the same performance as restricted queries. In particular, ULTRA-CSA is the first efficient multi-modal variant of CSA. Source code for ULTRA and our experiments is available at <https://github.com/kit-algo/ULTRA>.

2 Preliminaries

In this section we establish the basic notation and terminology used in this work. Moreover, we introduce the RAPTOR and Bucket-CH algorithms, on which our work is founded.

Public Transit Network. A public transit network is a 4-tuple $(\mathcal{S}, \mathcal{T}, \mathcal{R}, G)$ consisting of a set of *stops* \mathcal{S} , a set of *trips* \mathcal{T} , a set of *routes* \mathcal{R} , and a directed, weighted *transfer graph* $G = (\mathcal{V}, \mathcal{E})$. Every stop in \mathcal{S} defines a location in the network where passengers can board or disembark a vehicle (such as buses, trains, ferries, etc.). Furthermore, we associate with each stop $v \in \mathcal{S}$ a non-negative *departure buffer time* $\tau_{\text{buf}}(v)$, which defines the minimum amount of time that has to pass after arriving at the stop before a vehicle can be boarded. A trip $T = \langle v_0, \dots, v_k \rangle \in \mathcal{T}$ is a sequence of at least two stops which are served consecutively by the same vehicle. For each stop v in the sequence, $\tau_{\text{arr}}(T, v)$ denotes the *arrival time* of the vehicle at v , and $\tau_{\text{dep}}(T, v)$ denotes its *departure time*. This, of course, implies that $\tau_{\text{arr}}(T, v) \leq \tau_{\text{dep}}(T, v)$ holds for every trip T and stop v . The i -th stop of a trip T is denoted as $T[i]$. The set of routes \mathcal{R} defines a partition of the trips such that two trips are part of the same route if they have the same stop sequence and do not overtake each other. A trip $T_a \in \mathcal{T}$ overtakes the trip $T_b \in \mathcal{T}$ if two stops $u, v \in \mathcal{S}$ exist such that T_a arrives at or departs from u before T_b and T_a arrives at or departs from v after T_b .

The transfer graph $G = (\mathcal{V}, \mathcal{E})$ consists of a set of *vertices* \mathcal{V} with $\mathcal{S} \subseteq \mathcal{V}$, and a set of *edges* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. For each edge $e = (u, v) \in \mathcal{E}$ we define the *transfer time* $\tau_\theta(e)$ as the time required to transfer from u to v . The notion of transfer time carries over to paths $P = (v_1, \dots, v_k)$ in G , using the definition $\tau_\theta(P) := \sum_{i=1}^{k-1} (\tau_\theta(v_i, v_{i+1}))$. Unlike in restricted walking scenarios, we require no special properties for the transfer graph G . It does not need to be transitively closed, it may be strongly connected, and transfer times may represent walking, cycling, or some other non-schedule-based mode of travel.

Journeys. A *trip leg* T^{ij} is a subsequence of the trip T , representing a passenger boarding the trip T at the i -th stop and disembarking at the j -th stop. The departure time of T^{ij} is the departure time at the first stop of the trip leg, i.e., $\tau_{\text{dep}}(T^{ij}) := \tau_{\text{dep}}(T, T[i])$. Similarly, the arrival time is defined as $\tau_{\text{arr}}(T^{ij}) := \tau_{\text{arr}}(T, T[j])$. An *intermediate transfer* ϑ between two trip legs T_a^{ij} and T_b^{mn} is a path in the transfer graph such that: (1) the path ϑ begins with the last stop of T_a^{ij} , (2) the path ends with the first stop of T_b^{mn} , and (3) the transfer time of the path is sufficient to reach T_b^{mn} . The transfer time is sufficient if after

vacating T_a^{ij} and transferring to the departure stop of T_b^{mn} , there is still sufficient buffer time to enter T_b^{mn} . We can express this formally as $\tau_{\text{arr}}(T_a^{ij}) + \tau_{\theta}(\vartheta) + \tau_{\text{buf}}(T_b^{mn}[m]) \leq \tau_{\text{dep}}(T_b^{mn})$. An *initial transfer* ϑ before a trip leg T^{ij} is a path in G from the source s to the first stop of T^{ij} . Correspondingly, a *final transfer* ϑ after a trip leg T^{ij} is a path in G from the last stop of T^{ij} to the target t . We use the term *transfer* on its own to denote the union of all transfer types, or if the actual type of the transfer can be deduced from context. We define a *journey* $J = \langle \vartheta_0, T_0^{ij}, \dots, T_{k-1}^{mn}, \vartheta_k \rangle$ as an alternating sequence of transfers and trip legs. Note that some or all of the transfers may be empty. The departure time of the journey is defined as $\tau_{\text{dep}}(J) := \tau_{\text{dep}}(T_0^{ij}) - \tau_{\text{buf}}(T_0[i]) - \tau_{\theta}(\vartheta_0)$ and the arrival time as $\tau_{\text{arr}}(J) := \tau_{\text{arr}}(T_{k-1}^{mn}) + \tau_{\theta}(\vartheta_k)$. The number of trips used by the journey is k . A journey J (weakly) *dominates* a journey J' if $\tau_{\text{dep}}(J) \geq \tau_{\text{dep}}(J')$, $\tau_{\text{arr}}(J) \leq \tau_{\text{arr}}(J')$, and J does not use more trips than J' . For strict domination, at least one criterion must be strictly better. A journey J is called *Pareto-optimal* if no other journey exists that dominates J .

In our journey definition, the departure buffer time at a stop models the time required to reach the right platform and board a trip, regardless of how the stop was reached. Many other works on public transit routing instead use a *minimum transfer time*, which only needs to be observed if the stop was reached directly via a trip instead of a transfer. This is reasonable for settings with direct transfers between stops, where the buffer time can simply be included in the transfer time. When allowing arbitrary transfers, however, it can lead to inconsistencies. For instance, given a stop with minimum transfer time τ , if a path starting and ending at this stop with a transfer time less than τ exists, then taking that path would allow passengers to circumvent the minimum transfer time.

Algorithms. Since our algorithm is strongly influenced by the RAPTOR algorithm family, we now introduce the basic concepts of these algorithms. The RAPTOR [13] algorithm can be used to solve one-to-one and one-to-many queries on a public transit network with limited transfers. The algorithm operates in *rounds*, where the i -th round finds all journeys using exactly i trips. For this, each round extends journeys found in the previous round by one trip, which can be done via a single scan of all routes in the network. An extension of this algorithm for multi-modal scenarios with unlimited transfers is MCR [10]. In this algorithm the RAPTOR rounds are alternated with Dijkstra’s algorithm on a contracted transfer graph, in order to propagate arrival times through the transfer graph. Another extension, rRAPTOR [13], can be used to answer *range queries*, which ask for all Pareto-optimal journeys that depart within a given time interval. The rRAPTOR algorithm operates in *iterations*, where every iteration handles a possible departure time using the basic RAPTOR algorithm. The possible departure times are handled in descending order, and the data structures used by RAPTOR are not cleared in between iterations. As a result, journeys found by the current iteration are implicitly pruned by journeys that depart later and neither arrive later nor have more trips. This property of the rRAPTOR algorithm is called *self-pruning*.

Besides public transit routing algorithms, we also require efficient one-to-many algorithms for the transfer graph. Especially the Bucket-CH [21, 19, 20] algorithm is useful for our purposes. This algorithm is based on Contraction Hierarchies (CH) [19, 20] and operates in three phases. First the CH is computed, requiring only the graph. Second, given the set of targets, a bucket containing distances to the targets is computed for every vertex. This is done by adding every target to the buckets of all vertices in its reverse CH search space. Finally, the distance from a source to all targets is computed by performing the forward part of a CH search. For each vertex v in the forward search space, the bucket is evaluated by combining the distance to v with the distance from v to the targets in the bucket.

3 Shortcut Computation

Our preprocessing technique aims at finding a small number of transfer shortcuts that are sufficient to answer every point-to-point query correctly. This is achieved if for every Pareto-optimal journey there exists a journey with the same departure time, arrival time, and number of trips that uses only the precomputed shortcuts to transfer between trips. Next, we present a high-level overview of the ULTRA preprocessing, followed by an in-depth description of important algorithmic details.

3.1 Overview

The basic idea of ULTRA is as follows. We enumerate all possible journeys that use exactly two trips and require neither an initial nor a final transfer. The transfers between the two trips of these journeys are then considered as *candidates* for shortcuts. For each of these *candidate journeys*, we check if there is another journey that dominates it. If this is the case, we can replace the candidate journey with the dominating journey without losing Pareto-optimality. Note that if the candidate journey is contained in a longer journey, then it still can be replaced without affecting the Pareto-optimality of the longer journey. We call such a dominating journey a *witness* since its existence proves that the candidate shortcut is not needed. Unlike the candidate journey, the witness journey can make use of the transfer graph before the first trip or after the second trip. If no witness is found, then the candidate shortcut is added to the resulting shortcut graph.

A naive implementation of this idea would be to first enumerate all candidate journeys and subsequently search for witnesses. However, this would be impractical due to the sheer number of possible journeys. We therefore propose to interweave the candidate enumeration and the witness search, with the goal of eliminating as many candidates as early as possible. Pseudocode for the result of these considerations is given by Algorithm 1. The algorithm resembles invoking rRAPTOR [13] once per stop, restricted to the first two rounds per iteration. Remember that the original rRAPTOR algorithm already answers one-to-all range queries. Restricting this algorithm to the first two rounds enables an efficient enumeration of candidate journeys. Moreover, many dominated candidates are eliminated early on, due to self-pruning. We will now continue with a detailed discussion of Algorithm 1, showing step by step what has changed in comparison to the original rRAPTOR and how this helps with computing the transfer shortcuts.

3.2 Implementation Details

A first important difference is due to the fact that rRAPTOR requires a transitively closed transfer graph. As we want to allow arbitrary transfer graphs, we replace the RAPTOR that is invoked in every iteration of rRAPTOR with MR- ∞ , the variant of MCR that optimizes arrival time and number of used trips. Because of this change, the relaxation of transfers in lines 8 and 11 is not done by relaxing outgoing edges of updated stops. Instead, Dijkstra's algorithm is performed in order to propagate arrival times found by the preceding route scanning step. Furthermore, MCR would also use Dijkstra's algorithm in order to collect all routes reachable from the source stop in line 6. In the context of rRAPTOR this leads to many redundant computations, as the source stop does not change between iterations. We therefore compute distances from the source stop to all other stops once in line 3, again using Dijkstra's algorithm. These distances can then be used in line 6.

■ **Algorithm 1** ULTRA transfer shortcut computation.

Input: Public transit network $(\mathcal{S}, \mathcal{T}, \mathcal{R}, G)$, with unrestricted transfer graph $G = (\mathcal{V}, \mathcal{E})$
Output: Shortcut graph $G' = (\mathcal{S}, \mathcal{E}')$

```

1 for each  $s \in \mathcal{S}$  do
2   Clear all arrival labels and Dijkstra queues
3    $d(s, *) \leftarrow$  Compute distances from  $s$  to all stops in  $G$  using Dijkstra
4    $D \leftarrow$  Collect departure times of trips at  $s$ 
5   for each  $\tau_{dep} \in D$  in descending order do // rRAPTOR iteration
6     Collect routes reachable from  $s$  at  $\tau_{dep}$  // first RAPTOR round
7     Scan routes
8     Relax transfers
9     Collect routes serving updated stops // second RAPTOR round
10    Scan routes
11     $C \leftarrow$  Relax transfers, thereby collecting unwitnessed candidates
12     $\mathcal{E}' \leftarrow \mathcal{E}' \cup C$ 

```

Departure Time Collection. In line 4, standard rRAPTOR would collect all departure events that are reachable from the source stop s . However, given a transfer graph without any restrictions, this could possibly be every departure event in the network. Since we are primarily interested in finding candidate journeys, which do not have initial transfers, we collect only those departure events which depart directly at the source stop s . However, in order to find witness journeys, we still need to explore initial transfers in line 6. A naive implementation would check for each stop v reachable from s and for each route containing the stop v whether a trip that was not scanned in a previous iteration can be reached given the departure time τ_{dep} at s .

A more efficient approach combines lines 4 and 6 into a single operation. For this, we first sort all departure triplets (v, τ_{dep}, r) of departure stop v , departure time τ_{dep} , and route r by their corresponding departure time at the source, $\tau_{dep} - \tau_{buf}(v) - d(s, v)$. Afterwards, we iterate through this sorted list in descending order of departure time. If the next triplet to be processed has a departure stop $v \neq s$, then its route is added to a set \mathcal{R}' . In the case that the next triplet actually has the source stop s as departure stop v , we proceed with lines 6 through 12. Now the routes that have to be collected in line 6 are exactly the routes in \mathcal{R}' . Thus we simply scan all routes in \mathcal{R}' and then reset $\mathcal{R}' = \emptyset$ for the next iteration.

Limited Transfer Relaxation. Another part of ULTRA that differs from rRAPTOR is the final relaxation of transfers in line 11. This is the part of the algorithm where we actually determine the candidate journeys for which have not found a witness. As usual, relaxing the transfers is done by Dijkstra's algorithm, initialized with the arrival times from the preceding route scanning step. Whenever a stop is settled during this execution of Dijkstra's algorithm, we look at the corresponding journey and check whether it is a candidate journey, i.e., does not require initial or final transfers. If so, we know that there is no witness journey dominating this candidate, because otherwise the search would have reached the stop via this witness journey instead. Thus, we extract the intermediate transfer of the found candidate journey and add it as an edge to the shortcut graph.

We further increase the practical performance of our algorithm by adding a stopping criterion to the final transfer relaxation in line 11. For this purpose, we count the number of stops which were newly reached via a candidate journey in the preceding route scanning

step. Whenever such a stop is settled in line 11, we decrease our counter. Once the counter reaches zero, we can stop settling further vertices as we know that no more candidates can be found in this iteration. We can apply a similar stopping criterion to the intermediate transfer relaxation in line 8. In this case, we count the stops which were reached via a route directly from s , without an initial transfer, since only these stops can later become part of a candidate journey. As in line 11, we can stop settling vertices as soon as no such stops are left in the Dijkstra queue. This does not affect the correctness of the algorithm, as we still process all candidates. However, it might cause some witnesses to be pruned and thus lead to superfluous shortcuts in the result. To counteract this, we take the arrival time τ_{arr} of the last stop representing a candidate that is settled. Instead of stopping the transfer relaxation immediately, we continue until the queue head has an arrival time greater than $\tau_{\text{arr}} + \bar{\tau}$ for some parameter $\bar{\tau}$ (which we call *witness limit*). With these changes, the only remaining part of the algorithm that performs an unlimited search on the transfer graph is the initial transfer relaxation in line 3, which is only done once per source stop.

The success of our pruning rule for the transfer relaxation in lines 8 and 11 depends on the presence of candidate journeys in the Dijkstra queues. Fewer candidate journeys could therefore lead to an earlier application of the pruning rule. We exploit this by further restricting the notion of candidate journeys. As before, a candidate journey must not contain any initial or final transfers. In addition, we now require that the intermediate transfer of a candidate journey is not contained in the set of already computed transfer shortcuts.

Cyclic Witnessing. Since witnesses are only required to dominate candidate journeys weakly, there may be journeys J, J' that dominate each other. If J has an initial transfer of length > 0 , then J without the initial transfer is not dominated by J' extended by the reverse initial transfer. Therefore, the shortcut required by J will be added. Thus, cyclic domination is only problematic between journeys with initial transfers of length 0. We prevent this by temporarily contracting groups of stops with transfer distance 0 during the preprocessing.

Transfer Graph Contraction. As shown for MCR [10], the transfer relaxation is often the bottleneck of multi-modal routing algorithms. Since ULTRA only needs to compute journeys between stops, rather than arbitrary vertices of the transfer graph, only transfers that start and end at stops are relevant. Therefore, any overlay graph that preserves the distances between all stops can be used instead of the transfer graph in our preprocessing algorithm. An easy way of obtaining such an overlay graph is to construct a partial CH that only contracts vertices that do not correspond to stops of the public transit network. This, of course, leads to a suboptimal contraction order and thus makes it infeasible to contract all vertices that are not stops. As done in many other algorithms [6, 16, 10, 8, 7], we therefore stop the contraction once the uncontracted *core* graph surpasses a certain average vertex degree.

Parallelization. Finally, we observe that ULTRA allows for trivial parallelization. Our algorithm searches for candidate journeys once for every possible source stop (line 1 of Algorithm 1). As these searches are mostly independent of each other, we can distribute them to parallel threads and combine the results in a final sequential step. Only the usage of the restricted candidate notion introduces a dependence between the searches for different source stops. As this is only a heuristic performance optimization, we simply relax the notion of candidate journeys again, only requiring that no shortcut representing the intermediate transfer has been found by the same thread yet.

■ **Algorithm 2** Query algorithm, using transfer shortcuts computed by ULTRA.

Input: Public transit network $(\mathcal{S}, \mathcal{T}, \mathcal{R}, G)$, shortcut graph $G' = (\mathcal{S}, \mathcal{E})$,
 Bucket-CH of G , source vertex s , departure time τ_{dep} , and target vertex t

Output: All Pareto-optimal journeys from s to t for departure time τ_{dep}

- 1 $d(s, *) \leftarrow$ Run Bucket-CH query from s
- 2 $d(*, t) \leftarrow$ Run reverse Bucket-CH query from t
- 3 $G'' \leftarrow G'$
- 4 **for each** $v \in \mathcal{S}$ **do**
- 5 Add edge (s, v) to G'' with travel time $d(s, v)$
- 6 Add edge (v, t) to G'' with travel time $d(v, t)$
- 7 Run black box public transit algorithm on $(\mathcal{S} \cup \{s, t\}, \mathcal{T}, \mathcal{R}, G'')$

3.3 Proof of Correctness

Before continuing with the query algorithms, we want to justify that ULTRA computes a shortcut graph that is sufficient to answer all queries correctly. For contradiction we assume that a journey $J = \langle \vartheta_0, T_0^{ij}, \dots, T_{k-1}^{mn}, \vartheta_k \rangle$ exists that requires an intermediate transfer not contained in the shortcut graph and cannot be replaced with a journey of equal travel time and number of trips that solely uses transfers from the shortcut graph. In this case, the journey J must contain at least two trips, since otherwise it would not contain any intermediate transfers. Since the journey contains two or more trips, it can be disassembled into candidate journeys $\langle T_0^{ij}, \vartheta_1, T_1^{gh} \rangle, \langle T_1^{gh}, \vartheta_2, T_2^{pq} \rangle, \dots, \langle T_{k-2}^{uv}, \vartheta_{k-1}, T_{k-1}^{mn} \rangle$. As J requires a transfer not contained in the shortcut graph, at least one of these candidates must also contain a transfer not contained in the shortcut graph. Let $J^c = \langle T_x^{gh}, \vartheta_{x+1}, T_{x+1}^{pq} \rangle$ be such a candidate journey. Since the main loop of ULTRA is executed for every stop in the network, it was also executed for the source stop $T_x[g]$ of this candidate journey. Derived from the correctness of rRAPTOR, we know that for a given source stop our algorithm computes Pareto-optimal arrival labels for all stops reachable with two trips or less. Thus we also reached the target stop $T_{x+1}[q]$ of the candidate journey. The journey J' corresponding to the target's arrival label is in this case either the candidate journey or a journey that dominates the candidate journey. In the first case, we have added the transfer ϑ_{x+1} of the candidate journey to the shortcut graph. In the second case, the candidate journey J^c can be replaced by the journey J' corresponding to the target's arrival label, leading to a journey that is not worse than the original journey and does not require the missing transfer. Therefore both cases contradict our assumption.

4 Query Algorithms

The shortcuts obtained by ULTRA can in principle be combined with any public transit query algorithm that normally requires a transitively closed transfer graph, such as RAPTOR [13], CSA [14, 15], or Trip-Based Routing [27]. The basic idea of the query algorithm is to simply use one of the above algorithms together with our precomputed shortcut graph instead of the original transfer graph. However, our shortcut graph only represents transfers between two trips, and does not provide any information for transferring from the source to the first trip or from the last trip to the target. In this section we describe how the public transit algorithms can be modified in order to handle initial and final transfers efficiently.

Basic Query Algorithm. Our approach is based on the observation that for initial and final transfers one endpoint of the transfer is fixed. All initial transfers start at the source vertex, and all final transfers end at the target vertex. Therefore, we can use two additional one-to-many queries (one of them performed in reverse) to cover initial and final transfers. These queries have to be performed on the original transfer graph, where they compute the distances from the source to all stops and from all stops to the target. While any one-to-many algorithm might be used to perform this task, we decided to use Bucket-CH, as it is one of the fastest known one-to-many algorithms and allows for optimization of local queries. Pseudocode for the resulting query algorithm using Bucket-CH and our transfer shortcuts is shown in Algorithm 2.

Our algorithm begins with performing the two Bucket-CH queries from the source and target stop in lines 1 and 2. Afterwards a temporary copy of the shortcut graph G'' is initialized. In lines 5 and 6, this temporary graph is complemented with edges from the source to all other stops and edges from all stops to the target, using the distances obtained from the Bucket-CH queries. Finally, a public transit algorithm is invoked as a black box on the public transit network with the temporary graph instead of the shortcut graph in line 7. The temporary graph is sufficient for the query to yield correct results, as it contains edges from the source to any possible first stop, all edges required to transfer between trips, and edges from any possible last stop to the target. Since there are no additional requirements on the black box public transit algorithm, it is easy to see that any existing public transit algorithm can be used with our shortcuts.

Running Time Optimizations. We can further improve the performance of this query algorithm in practice by introducing some adjustments. First, we observe that we actually do not need edges from the source to every other stop. If the distance $d(s, v)$ from s to a stop v is greater than the distance $d(s, t)$ from s to t , every journey that requires a transfer from s to v is dominated by simply transferring directly from s to t . Thus, we do not need to add the edge (s, v) to the temporary graph. The same argument can be made for edges from some stop u to the target t if the distance $d(u, t)$ is greater than $d(s, t)$. Moreover, if we know that a stop v is further away from the source than the target, then we do not even need to compute the actual distance $d(s, v)$. We can use this fact to prune the search space of the Bucket-CH queries in lines 1 and 2. For this purpose, we first perform a standard bidirectional CH query from source to target that stops settling vertices from the forward (respectively backward) queue if the corresponding key is greater than the tentative distance from the source to the target. As a result we obtain the distance $d(s, t)$, as well as the partial forward (backward) CH search space from s (t), containing no vertices that have a greater distance from s (to t) than $d(s, t)$. We then perform the second phase of the Bucket-CH query (i.e., scanning the buckets) only for the vertices in the partial search spaces of the CH query. Furthermore, we store the entries in each bucket sorted by the distance to their target. Thus we can stop scanning through the bucket of a vertex u once we reach a stop v within the bucket with $d(s, u) + d(u, v) \geq d(s, t)$. Doing so can drastically improve local queries, as we do not need to look at all stops, but only at stops that are close to the source or target.

If we do not treat the underlying public transit algorithm as a black box, we can further improve practical performance by omitting the construction of the temporary graph G'' . Instead of adding edges from s to stops v , we can directly initialize the tentative arrival times used by most public transit algorithms with $\tau_{\text{dep}} + d(s, v)$. Instead of adding edges to t , we try to update the tentative arrival time at the target with the arrival time at v plus $d(v, t)$ whenever the arrival time at v is updated.

■ **Table 1** Sizes of the used public transit networks and their transfer graphs (full and transitive).

Network	Stops	Routes	Trips	Stop events	Vertices	Full edges	Tran. edges
Switzerland	25 426	13 934	369 534	4 740 929	604 167	1 847 140	4 687 016
Germany	244 055	231 089	2 387 297	48 495 169	6 872 105	21 372 360	22 645 480

5 Experiments

All algorithms were implemented in C++17 compiled with GCC version 7.3.1 and optimization flag `-O3`. All experiments were conducted on a machine with two 8-core Intel Xeon Skylake SP Gold 6144 CPUs clocked at 3.5 GHz, 192 GiB of DDR4-2666 RAM, and 24.75 MiB of L3 cache. The shortcut preprocessing was performed in parallel on all 16 cores. The transfer graph contraction and the queries were performed on a single core.

Networks. We evaluated our technique on the public transit networks of Switzerland and Germany, which were previously used in [26]. The Switzerland network was extracted from a publicly available GTFS feed¹ and consists of two successive business days (30th and 31st of May 2017). The Germany network is based on data from `bahn.de` for Winter 2011/2012, comprising two successive identical days. For both networks, stops and connections outside of the country borders were removed. As unrestricted transfer graphs, we used the road networks of Switzerland and Germany, including pedestrian zones and stairs, which were obtained from OpenStreetMap² data. Vertices with degree one and two were contracted unless they coincided with stops. Unless stated otherwise, we used walking as the transfer mode, assuming a walking speed of 4.5 km/h on each edge. To obtain transitively closed transfer graphs (for comparison with standard RAPTOR and CSA), we inserted an edge between all stops for which the distance in the transfer graph lies below a certain threshold (15 minutes for Switzerland, 8 minutes for Germany) and then computed the transitive closure. An overview of the networks is given in Table 1.

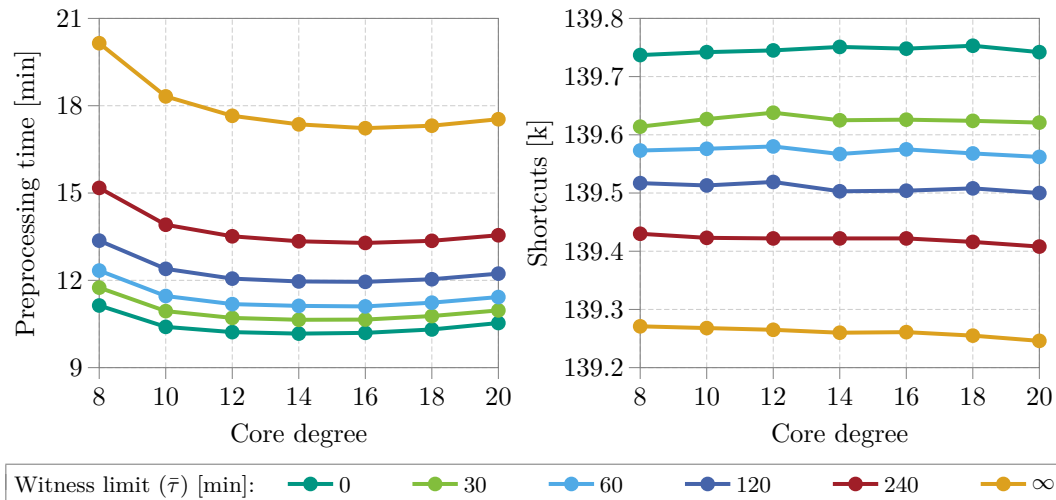
5.1 Preprocessing

In this section we evaluate the performance of the ULTRA preprocessing phase, including the transfer graph contraction and the shortcut computation.

Core Degree and Witness Limit. The two main parameters influencing the performance of the ULTRA preprocessing are the average vertex degree of the contracted transfer graph and the witness limit $\bar{\tau}$. Figure 1 shows the impact of these two parameters on the Switzerland network. The lowest preprocessing times are achieved with a core degree of 14. While the actual shortcut computation still becomes slightly faster for higher core degrees, this is offset by the increased time required to contract the transfer graph. Contracting up to a core degree of 14 took 1:29 minutes and yielded a graph with 32 683 vertices and 466 331 edges. By contrast, the witness limit $\bar{\tau}$ only has a minor impact on the number of computed shortcuts, with a difference of fewer than 600 shortcuts between $\bar{\tau} = 0$ and $\bar{\tau} = \infty$. For all following experiments, we chose a witness limit of 15 minutes, yielding 139 669 shortcuts for Switzerland.

¹ <http://gtfs.geops.ch/>

² <http://download.geofabrik.de/>

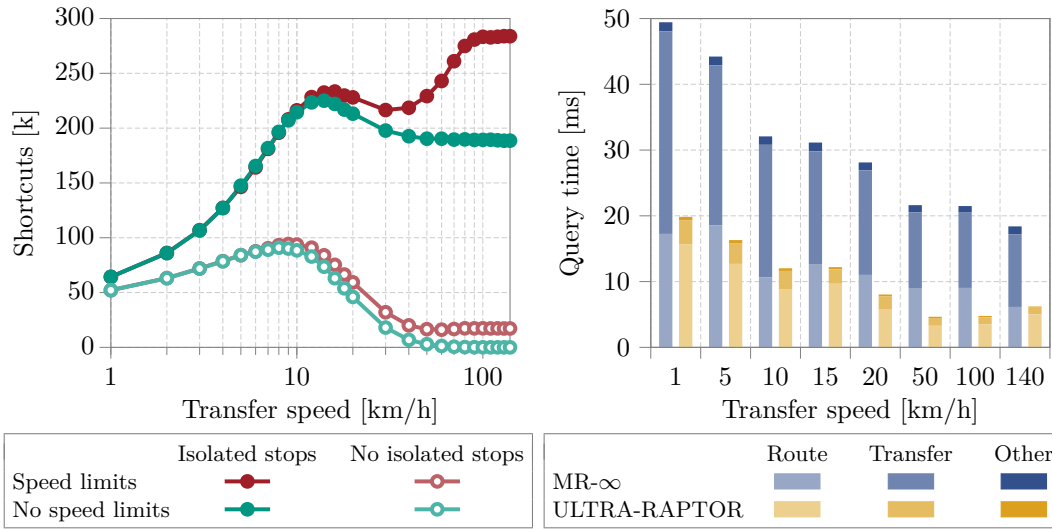


■ **Figure 1** Impact of core degree and witness limit on the running time of the preprocessing algorithm and the number of computed shortcuts, measured on the Switzerland network. Preprocessing time includes both contracting the transfer graph and computing the shortcuts.

For the Germany network, we chose to contract up to a core degree of 20, since the share of the core computation in the overall preprocessing time decreases as the network size increases. Contraction took 24:56 minutes and produced a core graph with 314 021 vertices and 6 280 440 edges. As before, we used a witness limit of 15 minutes for the shortcut computation, which yielded 2 077 374 shortcuts.

Parallelization. We used all 16 cores of our machine in parallel to accelerate the shortcut computation. On the Switzerland network this reduced the shortcut computation time from 2:02:55 hours sequentially to 9:35 minutes, which corresponds to a speedup of 12.8. Thus, we obtain a total preprocessing time of 11:05 minutes, including the time for the contraction, which was not parallelized. This yields an overall speedup for the preprocessing phase of 11.2. For the Germany network the sequential shortcut computation would take several days, while computing the shortcuts in parallel using all 16 cores took 10:53:35 hours.

Transfer Speed. In order to test the impact of the used transfer mode on the shortcut computation, we changed the transfer speed in the Switzerland network from 4.5 km/h to different values between 1 km/h and 140 km/h. We considered two ways of applying the transfer speed: In the first version, we did not allow the transfer speed on an edge to exceed the speed limit given in the road network. This allowed us to model fast transfer modes such as cars fairly realistically. In the second version, we ignored speed limits and assumed a constant speed on every edge. Thus, we can analyze to which extend the effects observed in the first version are caused by the speed limit data. Figure 2 (left side) reports the number of computed shortcuts measured for each configuration. In all measurements, the preprocessing time remained below 15 minutes. A peak in the number of shortcuts is reached between 10 and 20 km/h, which roughly corresponds to the speed of a bicycle. If speed limits are ignored, the number of shortcuts then starts decreasing again for higher transfer speeds and reaches a plateau at around 188 000 shortcuts. If speed limits are obeyed, the number of shortcuts eventually rises again and reaches the overall peak at 140 km/h, which was the highest speed limit observed in the network.



■ **Figure 2** Impact of transfer speed, measured on the Switzerland network with a core degree of 14 and a witness limit of 15 minutes. *Left:* Number of computed shortcuts. Speed limits in the network were obeyed for the red lines and ignored for the green lines. For the two lines at the bottom, shortcuts were only added to the result if the source and target stop for which they were found were connected by a path in the transfer graph. *Right:* Query performance of MR- ∞ and ULTRA-RAPTOR, averaged over 10 000 random queries. Speed limits were obeyed. Query times are divided into route collecting/scanning, transfer relaxation, and remaining time.

For low to medium transfer speeds, the results conformed with our expectations. As the transfer speed increases, it becomes increasingly feasible to cover large distances in the transfer graph quickly, making it possible to transfer between trips that are further away from each other. Accordingly, new shortcuts appear between these trips. However, once the transfer speed becomes competitive with the public transit vehicles, we would expect the number of shortcuts to decrease sharply as it eventually becomes preferable to avoid the public transit network altogether and transfer directly from source to target. The reason why this decrease is not observed in our measurements is that not all stops in our network instances are connected to the transfer graph. Consider what happens in the shortcut computation for journeys between stops s and t that are isolated from each other and the rest of the transfer graph. In this case, a direct transfer is not possible, regardless of the transfer speed. In fact, unless there is a route that serves both s and t , any optimal journey from s to t will include at least two trips. If a transfer is necessary between these two trips, then the journey is a nondominated candidate journey and a shortcut is added for the corresponding transfer. In our Switzerland network, 625 stops are isolated from the transfer graph, usually as a result of incomplete or imperfect data. To assess the impact of these stops on the number of computed shortcuts, we repeated our experiments, this time not adding shortcuts to the result if the source and target stop of the corresponding candidate journey were not connected in the transfer graph. This resulted in much fewer shortcuts, especially for high transfer speeds. If speed limits are ignored, the amount of necessary shortcuts becomes negligible at around 60 km/h and eventually reaches 0. If speed limits are obeyed, the number of shortcuts stagnates at 17 000.

Overall, these experiments show that our shortcut computation remains feasible regardless of the speed of the used transfer mode. Moreover, if the network does not include many stops that are isolated from the transfer graph, transferring between stops is most useful for transfer speeds between 10 and 20 km/h.

■ **Table 2** Query performance for RAPTOR, MR- ∞ , and ULTRA-RAPTOR. Query times are divided into phases: initial transfers, collecting routes, scanning routes, and relaxing transfers. All results are averaged over 10 000 random queries. RAPTOR (marked with *) only supports stop-to-stop queries with transitive transfers, instead of vertex-to-vertex queries on the full graph.

Network	Algorithm	Full graph	Scans [k]		Time [ms]				
			Routes	Edges	Init.	Collect	Scan	Relax	Total
Switzerland	RAPTOR*	○	27.2	3 527	0.0	3.7	6.4	7.8	18.4
	MR- ∞	●	34.9	769	11.6	5.9	8.2	12.3	39.3
	ULTRA-RAPTOR	●	37.7	148	1.6	4.9	7.9	1.9	16.7
Germany	RAPTOR*	○	480.4	25 798	0.0	166.9	178.0	85.1	436.5
	MR- ∞	●	555.8	12 571	191.1	250.7	202.2	272.2	944.1
	ULTRA-RAPTOR	●	610.6	2 224	26.8	204.5	202.9	37.0	477.8

5.2 ULTRA Queries

To evaluate the impact of our shortcuts on the query performance, we tested them with two public transit algorithms, RAPTOR and CSA. For each algorithm, we compared three query variants: one using our ULTRA approach, one using a transitively closed transfer graph, and one using a multi-modal variant of the algorithm on an unrestricted transfer graph.

RAPTOR Queries. In the case of RAPTOR, we used the MR- ∞ variant of MCR as the multi-modal algorithm, employing the same core graph that was used by the ULTRA pre-processing. The results of our comparison are shown in Table 2. Using ULTRA-RAPTOR drastically reduces the time consumption for exploring the transfer graph compared to MR- ∞ , from 50–60% of the overall running time to 10–20%. The reason for this is that both scanning the initial/final transfers and relaxing the intermediate transfers are an order of magnitude faster in ULTRA-RAPTOR compared to MR- ∞ . For the initial and final transfers, the Core-CH search of MR- ∞ is replaced by a Bucket-CH query in ULTRA-RAPTOR. Similarly, ULTRA-RAPTOR uses shortcuts for relaxing the intermediate transfers whereas MR- ∞ performs a Dijkstra search in the core graph. Overall, ULTRA-RAPTOR is twice as fast as MR- ∞ and has a similar running time to RAPTOR with transitive transfers. Note that comparing the running times of RAPTOR and ULTRA-RAPTOR has to be done with caution, as they were measured for a different set of queries. Hence, our shortcut technique enables RAPTOR to use unrestricted transfers without incurring the performance loss that is associated with MCR.

CSA Queries. For CSA, incorporating unrestricted transfers efficiently is more challenging. Since no multi-modal variant of CSA has been published thus far, we implemented a naive multi-modal version of CSA, which we call MCSA, as a baseline for our comparison. This algorithm alternates connection scans with Dijkstra searches on the contracted core graph, in a similar manner to MCR. Query times for all three CSA variants are reported in Table 3. Note that CSA solves an easier problem than RAPTOR, since it only minimizes the arrival time and not the number of transfers. When using a transitive transfer graph, it is thus approximately three times as fast as RAPTOR. With unrestricted transfers, we observe that MCSA is slower than MR- ∞ . This is because alternating between connection scans and Dijkstra searches causes MCSA to lose the main performance advantage of CSA,

■ **Table 3** Query performance for CSA, MCSA, and ULTRA-CSA. Query times are divided into two phases: initialization including initial transfers (Init.), and connection scans including intermediate transfers (Scan). All results are averaged over 10 000 random queries. CSA (marked with *) only supports stop-to-stop queries with transitive transfers, instead of vertex-to-vertex queries on the full graph.

Network	Algorithm	Full graph	Scans [k]		Time [ms]		
			Connections	Edges	Init.	Scan	Total
Switzerland	CSA*	○	126.7	1 307	0.2	5.0	5.1
	MCSA	●	88.0	5 337	12.9	48.4	61.3
	ULTRA-CSA	●	87.3	52	1.8	3.1	4.9
Germany	CSA*	○	2 620.3	6 216	2.9	162.1	165.1
	MCSA	●	1 568.2	118 026	233.6	1462.5	1696.1
	ULTRA-CSA	●	1 562.5	665	25.7	116.8	142.5

which is its high memory locality. When using ULTRA-CSA, however, this advantage is restored because only a few shortcut edges have to be relaxed after scanning each connection. Overall, ULTRA-CSA is only slightly slower than transitive CSA and about three times as fast as RAPTOR with shortcuts, making it the only efficient multi-modal variant of CSA known so far. Moreover, our query times for ULTRA-RAPTOR and ULTRA-CSA are significantly faster than those reported for the state-of-the-art techniques HLraptor and HLCSA [22], by a factor of 3.6 and 11.1, respectively. With respect to preprocessing time and space consumption, HL-based techniques are also outperformed by ULTRA.

In addition to overall performance, we also measured how query times for RAPTOR are impacted by the transfer speed. Results are shown in Figure 2 (right side). The performance gains for ULTRA-RAPTOR compared to MR- ∞ are similar for all transfer speeds, and in fact slightly better for higher speeds. In all cases, the entire query time for ULTRA-RAPTOR is similar to or lower than the time that MR- ∞ takes for the route scanning phases only.

6 Conclusion

We developed a technique which significantly speeds up the computation of Pareto-optimal journeys in a public transit network with an unrestricted transfer graph. We achieved this by efficiently computing shortcuts that provably represent all necessary transfers. Parallelization enables fast precomputation, taking a few minutes on the network of Switzerland. Our evaluation showed that the number of computed shortcuts is low, regardless of the underlying transfer mode. The shortcuts can be used without adjustments by any public transit algorithm that previously required a transitively closed transfer graph. For RAPTOR and CSA, we showed that using shortcuts leads to similar query times as using a transitively closed transfer graph. Consequently, shortcuts enable the computation of unrestricted multi-modal journeys without incurring the performance losses of existing multi-modal algorithms. In particular, combining shortcuts with CSA yields the first efficient multi-modal variant of CSA.

For future work, we would like to develop a shortcut-based query algorithm that can answer many-to-many queries. It would also be interesting to adapt our shortcut precomputation to scenarios with additional Pareto criteria, such as walking distance or cost. Furthermore, it should be possible to extend the ULTRA approach to more complicated transfer modes, including bike or car sharing.

References

- 1 Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks. In *International Symposium on Experimental Algorithms*, pages 230–241. Springer, 2011.
- 2 Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast Routing in Very Large Public Transportation Networks using Transfer Patterns. In *European Symposium on Algorithms*, pages 290–301. Springer, 2010.
- 3 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route Planning in Transportation Networks. In *Algorithm engineering*, pages 19–80. Springer, 2016.
- 4 Hannah Bast, Matthias Hertel, and Sabine Storandt. Scalable Transfer Patterns. In *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 15–29, 2016.
- 5 Hannah Bast and Sabine Storandt. Frequency-Based Search for Public Transit. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 13–22. ACM Press, November 2014.
- 6 Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner. Combining Hierarchical and Goal-directed Speed-up Techniques for Dijkstra’s Algorithm. *ACM Journal of Experimental Algorithmics*, 15:2.3:2.1–2.3:2.31, March 2010.
- 7 Moritz Baum, Valentin Buchhold, Julian Dibbelt, and Dorothea Wagner. Fast Exact Computation of Isochrones in Road Networks. In *International Symposium on Experimental Algorithms*, pages 17–32. Springer, 2016.
- 8 Moritz Baum, Julian Dibbelt, Andreas Gemsa, Dorothea Wagner, and Tobias Zündorf. Shortest Feasible Paths with Charging Stops for Battery Electric Vehicles. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL ’15, pages 44:1–44:10, New York, NY, USA, 2015. ACM.
- 9 Annabell Berger, Martin Grimmer, and Matthias Müller-Hannemann. Fully Dynamic Speed-Up Techniques for Multi-criteria Shortest Path Searches in Time-Dependent Networks. In *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA’10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 35–46. Springer, May 2010.
- 10 Daniel Delling, Julian Dibbelt, Thomas Pajor, Dorothea Wagner, and Renato Werneck. Computing Multimodal Journeys in Practice. In *International Symposium on Experimental Algorithms*, pages 260–271. Springer, 2013.
- 11 Daniel Delling, Julian Dibbelt, Thomas Pajor, and Renato Werneck. Public Transit Labeling. In *Proceedings of the 14th International Symposium on Experimental Algorithms (SEA’15)*, *Lecture Notes in Computer Science*, pages 273–285. Springer, 2015.
- 12 Daniel Delling, Julian Dibbelt, Thomas Pajor, and Tobias Zuendorf. Faster Transit Routing by Hyper Partitioning. In *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASIs)*, pages 8:1–8:14, Dagstuhl, Germany, 2017.
- 13 Daniel Delling, Thomas Pajor, and Renato F Werneck. Round-based Public Transit Routing. *Transportation Science*, 49(3):591–604, 2014.
- 14 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly Simple and Fast Transit Routing. In *International Symposium on Experimental Algorithms*, pages 43–54. Springer, 2013.
- 15 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Connection Scan Algorithm. *ACM Journal of Experimental Algorithmics*, 23:1.7:1–1.7:56, October 2018.
- 16 Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. User-Constrained Multimodal Route Planning. *ACM Journal of Experimental Algorithmics*, 19:3.2:1–3.2:19, April 2015.
- 17 Edsger W Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische mathematik*, 1(1):269–271, 1959.

- 18 Yann Disser, Matthias Müller-Hannemann, and Mathias Schnee. Multi-Criteria Shortest Paths in Time-Dependent Train Networks. In *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 347–361. Springer, June 2008.
- 19 Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 319–333. Springer, June 2008.
- 20 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 46(3):388–404, August 2012.
- 21 Sebastian Knopp, Peter Sanders, Dominik Schultes, Frank Schulz, and Dorothea Wagner. Computing Many-to-Many Shortest Paths Using Highway Hierarchies. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, pages 36–45. SIAM, 2007.
- 22 Duc-Minh Phan and Laurent Viennot. Fast Public Transit Routing with Unrestricted Walking through Hub Labeling. In *Proceedings of the Special Event on Analysis of Experimental Algorithms (SEA²)*, Lecture Notes in Computer Science. Springer, 2019.
- 23 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics*, 12(2.4):1–39, 2008.
- 24 Jonas Sauer. Faster Public Transit Routing with Unrestricted Walking. Master's thesis, Karlsruhe Institute of Technology, April 2018.
- 25 Ben Strasser and Dorothea Wagner. Connection Scan Accelerated. In *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 125–137. SIAM, 2014.
- 26 Dorothea Wagner and Tobias Zündorf. Public Transit Routing with Unrestricted Walking. In *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2017.
- 27 Sascha Witt. Trip-Based Public Transit Routing. In *Algorithms-ESA 2015*, pages 1025–1036. Springer, 2015.

Streaming and Massively Parallel Algorithms for Edge Coloring

Soheil Behnezhad

University of Maryland, College Park, MD, USA
soheil@cs.umd.edu

Mahsa Derakhshan

University of Maryland, College Park, MD, USA
mahsa@cs.umd.edu

MohammadTaghi Hajiaghayi

University of Maryland, College Park, MD, USA
hajiagha@cs.umd.edu

Marina Knittel

University of Maryland, College Park, MD, USA
mknittel@cs.umd.edu

Hamed Saleh

University of Maryland, College Park, MD, USA
hamed@cs.umd.edu

Abstract

A valid *edge-coloring* of a graph is an assignment of “colors” to its edges such that no two incident edges receive the same color. The goal is to find a proper coloring that uses few colors. (Note that the maximum degree, Δ , is a trivial lower bound.) In this paper, we revisit this fundamental problem in two models of computation specific to massive graphs, the *Massively Parallel Computations* (MPC) model and the *Graph Streaming* model:

Massively Parallel Computation. We give a randomized MPC algorithm that with high probability returns a $\Delta + \tilde{O}(\Delta^{3/4})$ edge coloring in $O(1)$ rounds using $O(n)$ space per machine and $O(m)$ total space. The space per machine can also be further improved to $n^{1-\Omega(1)}$ if $\Delta = n^{\Omega(1)}$. Our algorithm improves upon a previous result of Harvey et al. [SPAA 2018].

Graph Streaming. Since the output of edge-coloring is as large as its input, we consider a standard variant of the streaming model where the output is also reported in a streaming fashion. The main challenge is that the algorithm cannot “remember” all the reported edge colors, yet has to output a proper edge coloring using few colors.

We give a one-pass $\tilde{O}(n)$ -space streaming algorithm that always returns a valid coloring and uses 5.44Δ colors with high probability if the edges arrive in a random order. For adversarial order streams, we give another one-pass $\tilde{O}(n)$ -space algorithm that requires $O(\Delta^2)$ colors.

2012 ACM Subject Classification Theory of computation → Massively parallel algorithms; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Massively Parallel Computation, Streaming, Edge Coloring

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.15

Funding Supported in part by Guggenheim Fellowship, NSF grants CCF:SPX 1822738, IIS:BIGDATA 1546108, DARPA grant SI3CMD, UMD Year of Data Science Program Grant, and Northrop Grumman Faculty Award.

Acknowledgements We thank the anonymous reviewers for many helpful comments and suggestions.



© Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Marina Knittel, and Hamed Saleh;

licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 15; pp. 15:1–15:14



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Given a graph $G(V, E)$, an edge coloring of G is an assignment of “colors” to the edges in E such that no two incident edges receive the same color. The goal is to find an edge coloring that uses few colors. Edge coloring is among the most fundamental graph problems and has been studied in various models of computation, especially in distributed and parallel settings.

Denoting the maximum degree in the graph by Δ , it is easy to see that Δ colors are necessary in any proper edge coloring. On the other hand, Vizing’s celebrated theorem asserts that $\Delta + 1$ colors are always sufficient [39]. While determining whether a graph can be Δ colored is NP-hard, a $\Delta + 1$ coloring can be found in polynomial time [5, 21]. These algorithms are, however, highly sequential. As a result, in restricted settings, it is standard to consider more relaxed variants of the problem where more colors are allowed [2, 8, 20, 24, 25, 27, 29, 32, 34, 35, 36].

In this paper, we study edge coloring in large-scale graph settings. Specifically, we focus on the *Massively Parallel Computations* (MPC) model and the *Graph Streaming* model.

1.1 Massively Parallel Computation

The Model. The MPC model [10, 26, 31] is a popular abstraction of modern parallel frameworks such as MapReduce, Hadoop, Spark, etc. In this model, there are N machines, each with a space of S words¹ that all run in parallel. The input, which in our case is the edge-set of graph $G(V, E)$, is initially distributed among the machines arbitrarily. Afterwards, the system proceeds in synchronous rounds wherein the machines can perform any arbitrary local computation on their data and can also send messages to other machines. The messages are then delivered at the start of the next round so long as the total messages sent and received by each machine is $O(S)$ for local machine space S . The main parameters of interest are S and the round-complexity of the algorithm, i.e., the number of rounds it takes until the algorithm stops. Furthermore, the total available space over all machines should ideally be linear in the input size, i.e., $S \cdot N = O(|E|)$.

Related Work in MPC. We have seen a plethora of results on graph problems ever since the formalization of MPC. The studied problems include matching and vertex cover [1, 6, 14, 17, 22, 33, 12, 15], maximal independent set [22, 28, 12, 15], vertex coloring [7, 16, 28, 37, 38], as well as graph connectivity and related problems [3, 4, 13, 30, 9]. (This is by no means a complete list of the prior works.)

We have a good understanding of the complexity of vertex coloring in the MPC model, especially if the local space is near linear in n : Assadi et al. [7] gave a remarkable algorithm that using $\tilde{O}(n)$ space per machine, finds a $(\Delta + 1)$ vertex coloring in a constant number of rounds. The algorithm is based on a sparsification idea that reduces the number of edges from m to $O(n \log^2 n)$. But this algorithm alone cannot be used for coloring the edges, even if we consider the more relaxed $(2\Delta - 1)$ edge coloring problem which is equivalent to $(\Delta + 1)$ vertex coloring on the line graph. The reason is that the line-graph has $O(m)$ vertices where here m is the number of edges in the original graph. Therefore even after the sparsification step, we have $\tilde{O}(m)$ vertices in the graph which is much larger than the local space available in the machines.

¹ Throughout the paper, the stated space bounds are in the number of words that each denotes $O(\log n)$ bits.

Not much work has been done on the edge coloring problem in the MPC model. The only exception is the algorithm of Harvey et al. [28] which roughly works by random partitioning the *edges*, and then coloring each partition in a different machine using a sequential $(\Delta + 1)$ edge coloring algorithm. The choice of the number of partitions leads to a trade-off between the number of colors used and the space per machine required. The main shortcoming of this idea, however, is that if one desires a $\Delta + \tilde{O}(\Delta^{1-\Omega(1)})$ edge coloring, then a strongly super linear local space of $n\Delta^{\Omega(1)}$ is required.

Our main MPC result is the following algorithm which uses a more efficient partitioning. The key difference is that we use a *vertex* partitioning as opposed to the algorithm of Harvey et al. which partitions the edges.

► **Result 1** (Theorem 1). *There exists an MPC algorithm that using $O(n)$ space per machine and $O(m)$ total space, returns a $\Delta + \tilde{O}(\Delta^{3/4})$ edge coloring in $O(1)$ rounds.*

The algorithm exhibits a tradeoff between the space and the number of colors (see Theorem 1) and can be made more space-efficient as the maximum degree gets larger. For instance, if $\Delta > n^\epsilon$ for any constant $\epsilon > 0$, it requires a strictly sublinear space of $n^{1-\Omega(1)}$ to return a $\Delta + o(\Delta)$ edge coloring in $O(1)$ rounds. This is somewhat surprising since all previous non-trivial algorithms in the strictly sublinear regime of MPC require $\omega(1)$ rounds.

Our algorithm can also be implemented in $O(1)$ rounds of *Congested Clique*, leading to a $\Delta + \tilde{O}(\Delta^{3/4})$ edge coloring there. Prior to our work, no sublogarithmic round Congested Clique algorithm was known even for $(2\Delta - 1)$ edge coloring.

1.2 Streaming

The Model. In the standard graph streaming model, the edges of a graph arrive one by one and the algorithm has a space that is much smaller than the total number of edges. A particularly important choice of space is $\tilde{O}(n)$ – which is also known as the *semi-streaming* model [19] – so that the algorithm has enough space to store the vertices but not the edges. For edge coloring, the output is as large as the input, thus, we cannot hope to be able to store the output and report it in bulk at the end. For this, we consider a standard twist on the streaming model where the output is also reported in a streaming fashion. This model is referred to in the literature as the “W-streaming” model [18, 23]. We particularly focus on one-pass algorithms.

Designing one-pass W-streaming algorithms is particularly challenging since the algorithm cannot “remember” all the choices made so far (e.g., the reported edge colors). Therefore, even the sequential greedy algorithm for $(2\Delta - 1)$ edge coloring, which iterates over the edges in an arbitrary order and assigns an available color to each color upon visiting it, cannot be implemented since we are not aware of the colors used incident to an edge.

Our first result is to show that a natural algorithm w.h.p.² provides an $O(\Delta)$ edge coloring if the edges arrive in a random-order.

► **Result 2** (Theorem 9). *If the edges arrive in a random-order, there is a one-pass $\tilde{O}(n)$ space W-streaming edge coloring algorithm that always returns a valid edge coloring and w.h.p. uses $(2e + o(1))\Delta \approx 5.44\Delta$ colors.*

² Throughout, we use “w.h.p.” to abbreviate “with high probability” implying probability at least $1 - 1/\text{poly}(n)$.

15:4 Streaming and Massively Parallel Algorithms for Edge Coloring

If the edges arrive in an arbitrary order, we give another algorithm that requires more colors.

► **Result 3** (Theorem 10). *For any arbitrary arrival of edges, there is a one-pass $\tilde{O}(n)$ space W -streaming edge coloring algorithm that succeeds w.h.p. and uses $O(\Delta^2)$ colors.*

These are, to our knowledge, the first streaming algorithms for edge coloring.

2 The MPC Algorithm

In this section, we consider the edge coloring problem in the MPC model. Our main result in this section is an algorithm that achieves the following:

► **Theorem 1.** *For any parameter k (possibly dependent on Δ) such that $n/k \gg \log n$, there exists an MPC algorithm with $O(\frac{n\Delta}{k^2} + \frac{n}{k}\sqrt{\Delta \log n/k})$ space per machine and $O(m)$ total space that w.h.p. returns a $\Delta + O(\sqrt{k\Delta \log n})$ edge coloring in $O(1)$ rounds.*

By setting $k = \sqrt{\Delta} + \log n$, the space required per machine will be $O(n)$ and the number of colors would be $\Delta + \tilde{O}(\Delta^{3/4})$. Using a reduction from [11], this also leads to an $O(1)$ round Congested Clique algorithm using the same number of colors.

► **Corollary 2.** *There exists a randomized MPC algorithm with $O(n)$ local space, as well as a Congested Clique algorithm, that both w.h.p. find a $\Delta + \tilde{O}(\Delta^{3/4})$ edge coloring in $O(1)$ rounds.*

Moreover, assuming that $\Delta = n^{\Omega(1)}$, by setting $k = \Delta^{0.5+\epsilon}$ for a small enough constant $\epsilon \in (0, 1)$, we get the following $O(1)$ round algorithm which requires $n^{1-\Omega(1)}$ machine space, which is notably strictly sublinear in n :

► **Corollary 3.** *If $\Delta = n^{\Omega(1)}$, there exists a randomized MPC algorithm with $O(n/\Delta^{2\epsilon}) = n^{1-\Omega(1)}$ space per machine and $O(m)$ total space that w.h.p. returns a $\Delta + \tilde{O}(\Delta^{0.75+\epsilon/2})$ edge coloring in $O(1)$ rounds.*

The Idea Behind the Algorithm. The first step in the algorithm is a random partitioning of the *vertex set* into k groups, V_1, \dots, V_k . We then introduce one subgraph for each vertex subset, called G_1, \dots, G_k , and one subgraph for every pair of groups which we denote as $G_{1,2}, \dots, G_{1,k}, \dots, G_{k-1,k}$. Any such G_i is simply the induced subgraph of G on V_i . Moreover, any such $G_{i,j}$ is the subgraph on vertices $V_i \cup V_j$, with edges with one point in V_i and the other in V_j .

The general idea is to assign different *palettes*, i.e., subsets of colors, to different subgraphs so that the palettes assigned to any two neighboring subgraphs (i.e., those that share a vertex) are completely disjoint. A key insight to prevent this from blowing up the number of colors, is that since any two edges from $G_{i,j}$ and $G_{i',j'}$ with $i \neq i'$ and $j \neq j'$ cannot share endpoints by definition, it is safe to use the same color palette for them.

To assign these color palettes, we consider a complete k -vertex graph with each vertex v_i in it corresponding to partition V_i and each edge (v_i, v_j) in it corresponding to the subgraph $G_{i,j}$. We then find a k edge coloring of this complete graph, which exists by Vizing's theorem since maximum degree in it is $k - 1$. This edge coloring can actually be constructed extremely efficiently using merely the edges' endpoint IDs. Thereafter, we map each of these k colors to a color palette. By carefully choosing k and the number of colors in each palette, we ensure that: (1) The total number of colors required is close to Δ . (2) Each subgraph $G_{i,j}$ can be

properly edge-colored with those colors in its palette. (3) Each subgraph fits the memory of a single machine so that we can put it in whole there and run the sequential edge coloring algorithm on it.

■ **Algorithm 1** An MPC algorithm for edge coloring.

Parameter: k ;

Output: An edge coloring of a given graph $G = (V, E)$ with maximum degree Δ using $\Psi := \Delta + d\sqrt{k\Delta \log n}$ colors for some large enough constant d .

Independently and u.a.r. partition V into k subsets V_1, \dots, V_k ;

For every $i \in [k]$, let G_i be the induced subgraph of G on V_i ;

For every $i, j \in [k]$ with $i \neq j$, let $G_{i,j}$ be the subgraph of G including an edge $e \in E$ iff one end-point of e is in V_i and the other is in V_j ;

Partition $[\Psi]$ into $k+1$ disjoint subsets C_1, \dots, C_k, C' , which we call *color palettes*, in an arbitrarily way such that each palette has exactly $\frac{\Psi}{k+1}$ colors;

for each graph G_i *in parallel* **do**

| Color G_i sequentially in a single machine with palette C' ;

end

// In what follows, we implicitly construct a k edge coloring of a complete k -vertex graph K_k and assign palette C_α to subgraph $G_{i,j}$ where α is the color of edge (i, j) in K_k .

for each graph $G_{i,j}$ *in parallel* **do**

| Color $G_{i,j}$ sequentially in a machine with palette C_α where

$\alpha = ((i + j) \bmod k) + 1$;

end

The algorithm outlined above is formalized as Algorithm 1. We start by proving certain bounds on subgraphs' size and degrees.

▷ **Claim 4.** W.h.p., every subgraph of type G_i or $G_{i,j}$ has maximum degree $\frac{\Delta}{k} + O(\sqrt{\Delta \log \frac{n}{k}})$ and has at most $O(\frac{n\Delta}{k^2} + \frac{n}{k}\sqrt{\Delta \log n/k})$ edges.

Proof. Let us start with bounding the degree of an arbitrary vertex $v \in V_i$ in subgraph G_i . The degree of vertex v in G_i is precisely the number of its neighbors that are assigned to partition V_i . Since there are k partitions, the expected degree of v in G_i is $\deg_G(v)/k \leq \Delta/k$. Furthermore, since the assignment of vertices to the partitions is done independently and uniformly at random, by a simple application of Chernoff bound, v 's degree in G_i should be highly concentrated around its mean. Namely, with probability at least $1 - n^{-2}$, it holds that $\deg_{G_i}(v) \leq \frac{\Delta}{k} + O(\sqrt{\Delta \log n/k})$. Now, a union bound over the n vertices in the graph, proves that the degree of all vertices in their partitions should be at most $\frac{\Delta}{k} + O(\sqrt{\Delta \log n/k})$ with probability $1 - 1/n$.

Bounding vertex degrees in subgraphs of type $G_{i,j}$ also follows from essentially the same argument. The only difference is that we have to union bound over $n \cdot k$ choices, as we would like to bound the degree of any vertex v with say $v \in V_i$ in k subgraphs $G_{i,1}, \dots, G_{i,k}$. Nonetheless, since $k \leq n$, there are still $\text{poly}(n)$ many choices to union bound over. Thus, by changing the constants in the lower terms of the concentration bound, we can achieve the same high probability result.

Finally, we focus on the number of edges in each of the subgraphs. Each partition V_i has n/k vertices in expectation since the n vertices are partitioned into k groups independently and uniformly at random. A simple application of Chernoff and union bounds, implies that the number of vertices in each partition V_i is at most $O(\frac{n}{k})$ w.h.p., so long as $n/k \gg \log n$,

15:6 Streaming and Massively Parallel Algorithms for Edge Coloring

which is the case. Since the number of edges in each partition is less than the number of vertices times max degree, combined with the aforementioned bounds on the max degree, we can bound the number of edges in G_i and $G_{i,j}$ for any i and j by

$$O\left(\frac{n}{k}\right) \cdot O\left(\frac{\Delta}{k} + \sqrt{\frac{\Delta}{k} \log n}\right) = O\left(\frac{n\Delta}{k^2} + \frac{n}{k} \sqrt{\frac{\Delta}{k} \log n}\right),$$

which is the claimed bound. \triangleleft

Next, observe that we use palettes C_1, \dots, C_{k+1}, C' , each of size $\frac{\Psi}{k+1}$ to color the subgraphs. We need to argue that the maximum degree in each subgraph is at most $\frac{\Psi}{k+1} - 1$ to be able to argue that using Vizing's theorem in one machine, we can color any of the subgraphs with the assigned palettes. This can indeed be easily guaranteed if the constant d is large enough:

► **Observation 5.** *If constant d in Algorithm 1 is large enough, then maximum degree of every graph is at most $\frac{\Psi}{k+1} - 1$, w.h.p.*

Proof. We have $\Psi = \Delta + d\sqrt{k\Delta \log n}$ in Algorithm 1, therefore:

$$\frac{\Psi}{k+1} = \frac{\Delta}{k+1} + \frac{d\sqrt{k\Delta \log n}}{k+1} = \frac{\Delta}{k} + \Theta(\sqrt{\Delta \log n/k}),$$

where the hidden constants in the second term of the last equation can be made arbitrarily large depending on the choice of constant d . On the other hand, recall from Claim 4 that the maximum degree in any of the subgraphs is also at most $\frac{\Delta}{k} + O(\sqrt{\Delta \log n/k})$. Thus, the palette sizes are sufficient to color the subgraphs if d is a large enough constant. \blacktriangleleft

We are now ready to prove the algorithm's correctness.

► **Lemma 6.** *Algorithm 1 returns a proper edge coloring of G using $\Delta + O(\sqrt{k\Delta \log n})$ colors.*

Proof. The algorithm clearly uses $\Psi = \Delta + O(\sqrt{k\Delta \log n})$ colors, it remains to argue that the returned edge coloring is proper. Each subgraph (of type G_i or $G_{i,j}$) is sent to a single machine and edge-colored there using the palette that it is assigned to. Since by Observation 5, each palette has at least $\Delta' + 1$ colors for Δ' being the max degree in the subgraphs, there will be no conflicts in the colors associated to the edges within a partition. We only need to argue that two edges e and f sharing a vertex v that belong to two different subgraphs are not assigned the same color. Note that all subgraphs of type G_i are vertex disjoint and all receive the special color palette C' , thus there cannot be any conflict there. To complete the proof, it suffices to prove that any two subgraphs $G_{i,j}$ and $G_{i',j'}$ that share a vertex receive different palettes. Note that in this case, either $i = i'$ or $j = j'$ by the partitioning. Assume w.l.o.g. that $i = i'$ and thus $j \neq j'$. Based on Algorithm 1 for $G_{i,j}$ and $G_{i',j'}$ to be assigned the same color palette, it should hold that

$$((i+j) \bmod k) + 1 = ((i'+j') \bmod k) + 1.$$

Since $i = i'$, this would imply that $(j \bmod k) = (j' \bmod k)$, though this would not be possible given that both j and j' are in $[k]$ and that $j \neq j'$. Therefore, any two subgraphs that share a vertex receive different palettes and thus there cannot be any conflicts, completing the proof. \blacktriangleleft

Next, we turn to prove the space bounds.

► **Lemma 7** (Implementation and Space Complexity). *Algorithm 1 can be implemented with total space $O(m)$ and space per machine of $O(\frac{n\Delta}{k^2} + \frac{n}{k}\sqrt{\Delta \log n/k})$ w.h.p.*

Proof. We start with an implementation that uses the specified space per machine but can be wasteful in terms of the total space, then describe how we can overcome this problem and also achieve an optimal total space of $O(m)$.

We can use $k + \binom{k}{2}$ machines, each with a space of size $O(\frac{n\Delta}{k^2} + \frac{n}{k}\sqrt{\Delta \log n/k})$ to assign colors to the edges in parallel. The first m_1, \dots, m_k machines will be used for edge coloring on G_1, G_2, \dots, G_k respectively. The other $m_{k+1}, \dots, m_{k+\binom{k}{2}}$ machines will be used for edge coloring on the $G_{i,j}$ graphs. Lemma 4 already guarantees that each subgraph has size $O(\frac{n\Delta}{k^2} + \frac{n}{k}\sqrt{\Delta \log n/k})$ w.h.p., and thus fits the memory of a single machine.

In the implementation discussed above, since the machines use $\tilde{O}(n\Delta/k^2)$ space and there are $O(k^2)$ machines, the total memory can be $\tilde{O}(n\Delta)$ which may be much larger than $O(m)$. This is because we allocate $O(n\Delta/k^2)$ space to each machine regardless of how much data it actually received. Though, observe that each edge of the graph belongs to exactly one of the subgraphs, i.e., the machines together only handle a total of $O(m)$ data. So we must consolidate into fewer machines. We do this by putting multiple subgraphs in each machine.

We start by recalling a sorting primitive in the MPC model which was proved in [26]. Basically, if there are N items to be sorted and the space per machine is $N^{\Omega(1)}$, then the algorithm of [26] sorts these items into the machines within $O(1)$ rounds. To use this primitive, we first label each edge $e = (u, v)$ of the graph by its subgraph name (e.g. G_i or $G_{i,j}$) which can be determined solely based on the end-points of the edge. After that, we sort the edges based on these labels. This way, all the edges inside each subgraph can be sent to the same machine within $O(1)$ rounds while also ensuring that the total required space remains $O(m)$. ◀

The algorithm for Theorem 1 was formalized as Algorithm 1. We showed in Lemma 6 that the algorithm correctly finds an edge coloring of the graph with the claimed number of colors. We also showed in Lemma 7 that the algorithm can be implemented with $O(m)$ total space and $O(\frac{n\Delta}{k^2} + \frac{n}{k}\sqrt{\Delta \log n/k})$ space per machine. This completes the proof of Theorem 1.

3 Streaming Algorithms

We start in Section 3.1 by describing our streaming algorithm and its analysis when the arrival order is random. Then in Section 3.2, we give another algorithm for adversarial order streams.

3.1 Random Edge Arrival Setting

In this section, we give a streaming algorithm for $O(\Delta)$ edge coloring using $\tilde{O}(n)$ space where the edges come in a random stream. That is, a permutation over the edges is chosen uniformly at random and then the edges arrive according to this permutation.

We first note that if $\Delta = O(\log n)$ then the problem is trivial as we can store the whole graph and then report a $\Delta + 1$ edge coloring (even without knowledge of Δ). As such, we assume $\Delta = \omega(\log n)$.

The algorithm – formalized as Algorithm 2 – maintains a counter c_v for each vertex v . At any point during the algorithm, this counter c_v basically denotes the highest color number used for the edges incident to v so far, plus 1. Therefore, upon arrival of an edge (u, v) , it is safe to color this edge with $\max(c_u, c_v)$ as all edges incident to u and v have a color that is

strictly smaller than this. Then, we increase the counters of both v and u to $\max(c_u, c_v) + 1$. It is not hard to see that the solution is always a valid coloring, in the remainder of this section, we mainly focus on the number of colors required by this algorithm and show that w.h.p., it is only $O(\Delta)$ for random arrivals.

■ **Algorithm 2** Edge coloring for random streams.

Result: A feasible coloring $\mathcal{C} : E \rightarrow [\Psi]$ for a given graph $G = (V, E)$ with maximum degree Δ in a random stream

$c_v \leftarrow 0 \quad \forall v \in V;$

while (u, v) is read from stream **do**

$\mathcal{C}(u, v) \leftarrow \max(c_u, c_v);$

$c_u, c_v \leftarrow \mathcal{C}(u, v) + 1;$

end

We start by noting that this algorithm can actually be extremely bad if the order is adversarial. To see this, consider a path of size n . In an adversarial stream where the edges arrive in the order of the path, Algorithm 2 uses as many as $n - 1$ colors while the maximum degree is only 2! It is easy to see why this example is very unlikely to occur in random order streams: For a fixed path, it is very unlikely that the edges are randomly ordered in this very specific way.

To make this intuition rigorous for general graphs, we first prove the following crucial lemma which gives us the correct parameter to bound.

► **Lemma 8.** *Let Ψ be the size of the longest monotone (in the order of arrival) path in the line-graph of G . Then Algorithm 2 uses exactly Ψ colors.*

Proof. Take a monotone path v_1, v_2, \dots, v_Ψ in the line-graph of G and let e_1, e_2, \dots, e_Ψ be the edges of the original graph that correspond to these vertices respectively, i.e., e_1 arrives before e_2 which arrives before e_3 and so on. Since for any i , v_i and v_{i+1} are neighbors in the line-graph, then e_i and e_{i+1} should share an end-point v . This means that at the time of arrival of e_{i+1} , we have $c_v \geq \mathcal{C}(e_i) + 1$ which in turn, implies $\mathcal{C}(e_\Psi) > \mathcal{C}(e_{\Psi-1}) > \dots > \mathcal{C}(e_1)$. Therefore, $\mathcal{C}(e_\Psi) \geq \Psi$.

On the other hand, suppose that there is an edge $e_1 = (u, v)$ for which $\mathcal{C}(e_1) = \Psi$ in Algorithm 2. This means that at least one of c_u or c_v equals Ψ when e_1 arrives, say c_u w.l.o.g. Let e_2 be the last edge incident to u that has arrived before e_1 . It should hold that $\mathcal{C}(e_2) = \Psi - 1$. Using the same argument, for each $1 < i \leq \Psi$, we can find a neighboring edge e_i such that $\mathcal{C}(e_i) = \mathcal{C}(e_{i-1}) - 1$. This way, we end up with a sequence e_1, \dots, e_Ψ of edges, the path corresponding to this sequence in the line graph will be a monotone path of length Ψ , completing the proof. ◀

► **Theorem 9.** *There is a streaming edge coloring algorithm that for any graph $G = (V, E)$ uses at most $(2e + \epsilon)\Delta \approx 5.44\Delta$ colors w.h.p. for any constant $\epsilon > 0$ given that the edges in E arrive in a random order.*

Proof. We first prove that Algorithm 2 gives us a feasible coloring of graph G . Consider two edges $e_1 = (u, v)$ and $e_2 = (u, v')$ incident to vertex u such that e_1 appears earlier than e_2 in the stream. For any edge e we represent by $\mathcal{C}(e)$ the color assigned to that by the algorithm. After the algorithm colors e_1 with $\mathcal{C}(e_1)$, it sets c_u to $\mathcal{C}(e_1) + 1$. Thus, c_u is at least $\mathcal{C}(e_1) + 1$ when e_2 arrives and $\mathcal{C}(e_2) \geq \mathcal{C}(e_1) + 1$ consequently. Therefore, $\mathcal{C}(e_2) > \mathcal{C}(e_1)$ for any pair of edges incident to a common vertex, and \mathcal{C} is a feasible coloring.

Next, for some constant α that we fix later, we show that the probability that an edge is assigned a color number at least $\alpha\Delta$ is at most n^{-c} for some constant $c \geq 2$, implying via a union bound over all the edges that indeed w.h.p., $\Psi \leq \alpha\Delta$.

We showed in Lemma 8 that if the number of colors Ψ used is $\alpha\Delta$, then there should exist a monotone path in the line-graph with size at least $\alpha\Delta$. Let $e_0, e_2, \dots, e_{\alpha\Delta}$ be the corresponding edges to this path. Thus, it suffices to bound the probability of this event. Let Π denote the set of all such paths in the line graph. For a specific path $\pi \in \Pi$, the probability that it is monotone is $1/(\alpha\Delta)!$. Call this event X_π . On the other hand, we can upper bound the number of such paths by $(2\Delta)^{\alpha\Delta}$, i.e., $|\Pi| \leq (2\Delta)^{\alpha\Delta}$. This follows from the fact that each path should start from the corresponding vertex to e_0 in the line-graph, and that maximum degree in the line graph is $2\Delta - 2$ (which is the upper bound on the number of neighboring edges to each edge). Thus:

$$\Pr[\mathcal{C}(e_0) \geq \alpha\Delta] = \Pr\left[\bigvee_{\pi \in \Pi} X_\pi\right] \leq \sum_{\pi \in \Pi} \Pr[X_\pi = 1] \leq \frac{(2\Delta)^{\alpha\Delta}}{(\alpha\Delta)!},$$

where the last inequality is obtained by replacing $\Pr[X_\pi = 1]$ and $|\Pi|$ by the aforementioned bounds. Taking the logarithm of each side of the inequality, we get

$$\begin{aligned} \ln(\Pr[\mathcal{C}(e_0) \geq \alpha\Delta]) &\leq \alpha\Delta \ln(2\Delta) - \ln((\alpha\Delta)!) \\ &\leq \alpha\Delta \ln(2\Delta) - ((\alpha\Delta + 1/2) \ln(\alpha\Delta) - \alpha\Delta) & (1) \\ &= \alpha\Delta \ln(2e/\alpha) - 1/2 \ln(\alpha\Delta) & (2) \\ &\leq \alpha\Delta \ln(2e/\alpha). & (3) \end{aligned}$$

To obtain (1), we use Stirling's approximation of factorials to lower-bound $\ln((\alpha\Delta)!)$. Finally, we rearranged terms to imply (2). By plugging in $\alpha = 2e(1 + \epsilon)$, we get

$$\begin{aligned} \ln(\Pr[\mathcal{C}(e_0) \geq 2e(1 + \epsilon)\Delta]) &\leq 2e(1 + \epsilon)\Delta \ln\left(\frac{1}{1 + \epsilon}\right) \\ &= -2e(1 + \epsilon) \ln(1 + \epsilon)\Delta \\ &\leq -2e(1 + \epsilon) \ln(1 + \epsilon) \frac{c}{2e(1 + \epsilon) \ln(1 + \epsilon)} \ln(n) & (4) \\ &= -c \ln(n) \end{aligned}$$

Since $\Delta = \omega(\log(n))$, we have $\Delta > c' \ln(n)$ for any constant c' . Inequality (4) follows from setting $c' = c/(2e(1 + \epsilon) \ln(1 + \epsilon))$ in $\Delta > c' \ln(n)$, where c is the constant for which we want to show the probability is upper-bounded by n^{-c} . Hence,

$$\Pr[\mathcal{C}(e_0) \geq 2e(1 + \epsilon)\Delta] \leq n^{-c}.$$

Thus, Algorithm 2 returns a feasible coloring of the input graph G using at most $2e(1 + \epsilon)\Delta$ colors, for any constant $\epsilon > 0$ w.h.p. if the edges arrive in a random order. \blacktriangleleft

To further evaluate the performance of Algorithm 2, we implemented and ran it for cliques of different size. The result of this experiment is provided in Table 1. The numbers are obtained by running the experiment 100 times and taking the average number of colors used. As it can be observed from Table 1, for cliques of size 100 to 1000, the number of colors used by the algorithm is in range $[3.3\Delta, 3.9\Delta]$ and it slightly increases by the size of the graph. Our analysis, however, shows that it should never exceed 5.44Δ .

15:10 Streaming and Massively Parallel Algorithms for Edge Coloring

■ **Table 1** The number of colors used by Algorithm 2 on cliques averaged over 100 trials.

Clique Size	100	200	300	400	500	600	700	800	900	1000
Colors Used	3.363 Δ	3.563 Δ	3.665 Δ	3.717 Δ	3.756 Δ	3.787 Δ	3.815 Δ	3.838 Δ	3.849 Δ	3.863 Δ

3.2 Adversarial Edge Arrival Setting

In this section, we turn to arbitrary (i.e., adversarial) arrivals of the edges. We assume that the adversary is *oblivious*, i.e., the order of the edges is determined before the algorithm starts to operate so that the adversary cannot abuse the random bits used by the algorithm. Having this assumption, we give a randomized algorithm that w.h.p., outputs a valid edge coloring of the graph using $O(\Delta^2)$ colors while using $\tilde{O}(n)$ space. The algorithm is formalized as Algorithm 3. We note that this algorithm, as stated, requires knowledge of Δ . However we later show that we can get rid of this assumption. Overall, we get the following result:

■ **Algorithm 3** Edge coloring in the adversarial order.

Result: A feasible coloring for a given graph $G = (V, E)$ with maximum degree Δ

```

for any vertex  $v \in V$  do
   $r_v \leftarrow$  a sequence of  $\log(n)$  independent random bits.
  for any  $i \in [\log n]$  do
     $c_{v,i} \leftarrow 0$ 
  end
end
for any edge  $e = (u, v)$  in the stream do
  Let  $i$  be the smallest index for which  $r_{v,i} \neq r_{u,i}$ .
  if  $\Delta 2^{-i} > \log n$  then
    if  $r_{u,i} = 1$  then
      Assign color  $(c_{u,i}, c_{v,i}, i)$  to  $e$ .
    else
      Assign color  $(c_{v,i}, c_{u,i}, i)$  to  $e$ .
    end
    Increase both  $c_{v,i}$  and  $c_{u,i}$  by one.
  else
    Store edge  $e$ .
  end
end
Color the stored edges using a new set of colors.

```

► **Theorem 10.** *Given a graph G with maximum degree Δ , there exists a one pass streaming algorithm, that outputs a valid edge coloring of the G using $O(\Delta^2)$ colors w.h.p., using $\tilde{O}(n)$ memory.*

Consider two vertices v and u and their string of random bits r_v and r_u defined in the algorithm. Let $d_{u,v}$ be the smallest index i where $r_{u,i} \neq r_{v,i}$. Upon arrival of an edge $e = (u, v)$, we first find $i := d_{u,v}$. If $\Delta 2^{-i} > \log n$, we color the edge immediately. Otherwise, we store it. We will show that all the stored edges fit in the memory thus after reading all the stream we can color them with a palette of at most $\Delta + 1$ new colors. In the algorithm, for any vertex v and any $i \in [\log n]$, we define a counter $c_{u,i}$. If $\Delta 2^{-i} > \log n$ for any edge e , then we immediately assign e a color which is represented by a tuple $(c_{u,i}, c_{v,i}, i)$. Then,

we increase counters $c_{u,i}$ and $c_{v,i}$. Note that we say two colors are the same if all three elements of them are equal. We first show that this gives us a valid coloring, which means it does not assign the same color to two edges adjacent to the same vertex. We use proof by contradiction. Assume that our algorithm assigns the same color to edges $e_1 = (u, v_1)$ and $e_2 = (u, v_2)$ adjacent to vertex u . None of them can be from the stored edges since we color them using a new palette. This means that $d_{u,v_1} = d_{u,v_2}$. Let us denote it by i . Without loss of generality, we assume that $r_{u,i} = 1$ and that in the input stream e_1 arrives before e_2 . Note that the first element of the colors (which are tuples) assigned to these edges is the value of counter $c_{u,i}$ when they arrive. However, the algorithm increases $c_{u,i}$ by one after arrival of e_1 thus the colors assigned to e_1 and e_2 cannot be the same.

Now, it suffices to show that the total number of colors used by the algorithm is $O(\Delta^2)$. Given a vertex v , and a number $l \in [\log n]$ let us compute an upper-bound for counter $c_{v,i}$. Let N_v be the set of neighbors of this vertex and let $N_{v,i}$ be the set of neighbors like u where $d_{v,u} = i$. We know that $c_{v,i} = |N_{v,i}|$, thus given any vertex v and $i \in [\log(n)]$, we need to find a bound for $|N_{v,i}|$. Given any edge $e = (v, u)$ the probability of e being in set $N_{v,i}$ is 2^{-i} which means $\mathbb{E}[|N_{v,i}|] = \deg(v)2^{-i}$ where $\deg(v)$ is the degree of vertex v in the input graph.

Using a simple application of the Chernoff bound, for any vertex v , we get:

$$\Pr\left[|N_{v,i}| \geq \deg(v)2^{-i} + O(\sqrt{\deg(v)2^{-i} \log n})\right] \leq \frac{1}{n^c}.$$

Setting c to be a large enough constant, one can use union bound and show that w.h.p., for any vertex v and $i \in [\log n]$ where $\deg(v)2^{-i} \geq \log n$, we have $|N_{v,i}| \leq O(\deg(v)2^{-i})$.

Having this, we conclude that for any $i \in [\log n]$, where $\Delta 2^{-i} > \log n$, the number of colors used by the algorithm whose third element is i is at most $O(\Delta^2 2^{-2i})$ since the first and the second element of the color can get at most $O(\Delta 2^{-i})$ different values. Therefore, the total number of colors used for any such i is at most $O(\sum_{i \in [\log n]} \Delta^2 2^{-2i}) = O(\Delta^2)$. We will also show that the stored edges fit in the memory and thus we can color them using $O(\Delta)$ new colors. As a result the total number of colors used is $O(\Delta^2)$.

To give an upper-bound for the number of stored edges we first show that the expected number of stored edges for each vertex is $O(\log n)$. Let $j := \log(\frac{\Delta}{\log n})$. Recall that we store an edge (u, v) when $\Delta 2^{-d_{u,v}} < \log n$. Thus the expected number of stored edges adjacent to a single vertex v is at most

$$\sum_{j \leq i \leq \log n} d_v 2^{-i} \leq \sum_{j \leq i \leq \log n} \Delta 2^{-i} \leq \sum_{j \leq i \leq \log n} \log(n) 2^{-i+j} = O(\log n).$$

To get the last equation we use the fact that $\Delta 2^{-j} \leq \log n$. By a similar argument that we used above (using Chernoff and Union bounds), with a high probability the total number of stored edges is $O(n \log n)$ which can be stored in the memory. Therefore the proof of this theorem is completed.

Knowledge of Δ . As written, our algorithm depends on the knowledge of Δ because we must check $\Delta 2^{-i} > \log n$. We can get rid of this condition by keeping track of the degree \deg_v^H of a vertex in the subgraph H we have seen so far, and then computing the max degree \deg_{max}^H . This only requires an additional $O(n)$ space. Thereafter, instead of checking if $\Delta 2^{-i} > \log n$, we check if $\deg_{max}^H 2^{-i} > \log n$. Whenever \deg_{max}^H increases, we iterate over all stored edges and recompute whether or not $\deg_{max}^H 2^{-i} > \log n$. If so, we color the edge and remove it from the buffer, else we keep it. It is easy to see that this will not exceed the space bounds because at any timestep, we can assume the input graph was H in the first

place. Then its max degree is $\Delta_H = \deg_{\max}^H$, and we can apply the same argument for the space bounds as before, but using Δ_H instead of Δ . All other parts of the proof still hold. Therefore our algorithm does not require knowledge of Δ .

Finally, we remark that if one allows more space, then one can modify Algorithm 3 to use fewer number of colors. Though we focused only on the $\tilde{O}(n)$ memory regime.

4 Open Problems

We believe the most notable future direction is to improve the number of colors used in our streaming algorithms. Specifically, our streaming algorithm for adversarial arrivals requires $O(\Delta^2)$ colors. A major open question is whether this can be improved to $O(\Delta)$ while also keeping the memory near-linear in n . Also for random arrival streams, we showed that Algorithm 2 achieves a 5.44Δ coloring and showed, experimentally, that it uses at least 3.86Δ colors. A particularly interesting open question is whether there is an algorithm that uses arbitrarily close to 2Δ colors using $\tilde{O}(n)$ space in random arrival streams.

References

- 1 Kook Jin Ahn and Sudipto Guha. Access to Data and Number of Iterations: Dual Primal Algorithms for Maximum Matching under Resource Constraints. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*, pages 202–211, 2015. doi:10.1145/2755573.2755586.
- 2 Noga Alon, László Babai, and Alon Itai. A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem. *J. Algorithms*, 7(4):567–583, December 1986. doi:10.1016/0196-6774(86)90019-2.
- 3 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. doi:10.1145/2591796.2591805.
- 4 Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. Parallel Graph Connectivity in Log Diameter Rounds. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 674–685, 2018. doi:10.1109/FOCS.2018.00070.
- 5 Eshrat Arjomandi. An efficient algorithm for colouring the edges of a graph with $\Delta + 1$ colours. *INFOR: Information Systems and Operational Research*, 20(2):82–101, 1982.
- 6 Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. Coresets Meet EDCS: Algorithms for Matching and Vertex Cover on Massive Graphs. *Proceedings of the 30th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, to appear, 2019.
- 7 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear Algorithms for $(\Delta + 1)$ Vertex Coloring. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 767–786, 2019. doi:10.1137/1.9781611975482.48.
- 8 Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The Locality of Distributed Symmetry Breaking. *J. ACM*, 63(3):20:1–20:45, June 2016. doi:10.1145/2903137.
- 9 MohammadHossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Raimondas Kiveris, Silvio Lattanzi, and Vahab S. Mirrokni. Affinity Clustering: Hierarchical Clustering at Scale. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6867–6877, 2017. URL: <http://papers.nips.cc/paper/7262-affinity-clustering-hierarchical-clustering-at-scale>.

- 10 Paul Beame, Paraschos Koutris, and Dan Suci. Communication Steps for Parallel Query Processing. *J. ACM*, 64(6):40:1–40:58, 2017. doi:10.1145/3125644.
- 11 Soheil Behnezhad, Mahsa Derakhshan, and MohammadTaghi Hajiaghayi. Brief Announcement: Semi-MapReduce Meets Congested Clique. *CoRR*, abs/1802.10297, 2018. arXiv:1802.10297.
- 12 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, and Richard M. Karp. Massively Parallel Symmetry Breaking on Sparse Graphs: MIS and Maximal Matching. *CoRR*, abs/1807.06701, 2018. arXiv:1807.06701.
- 13 Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, and Vahab Mirrokni. Near-Optimal Massively Parallel Graph Connectivity. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, to appear*, 2019.
- 14 Soheil Behnezhad, MohammadTaghi Hajiaghayi, and David G. Harris. Exponentially Faster Massively Parallel Maximal Matching. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, to appear*, 2019.
- 15 Sebastian Brandt, Manuela Fischer, and Jara Uitto. Matching and MIS for Uniformly Sparse Graphs in the Low-Memory MPC Model. *CoRR*, abs/1807.05374, 2018. arXiv:1807.05374.
- 16 Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The Complexity of $(\Delta + 1)$ Coloring in Congested Clique, Massively Parallel Computation, and Centralized Local Computation. *CoRR*, abs/1808.08419, 2018. arXiv:1808.08419.
- 17 Artur Czumaj, Jakub Lacki, Aleksander Madry, Slobodan Mitrovic, Krzysztof Onak, and Piotr Sankowski. Round Compression for Parallel Matching Algorithms. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 471–484, 2018. doi:10.1145/3188745.3188764.
- 18 Camil Demetrescu, Irene Finocchi, and Andrea Ribichini. Trading off space for passes in graph streaming problems. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 714–723, 2006. URL: <http://dl.acm.org/citation.cfm?id=1109557.1109635>.
- 19 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On Graph Problems in a Semi-streaming Model. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, pages 531–543, 2004. doi:10.1007/978-3-540-27836-8_46.
- 20 Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local Conflict Coloring. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 625–634, 2016. doi:10.1109/FOCS.2016.73.
- 21 H. N. Gabow, T. Nishizeki, O. Kariv, D. Leven, , and O. Terada. Algorithms for edgecoloring graphs. Technical report, Tohoku University, 1985.
- 22 Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrovic, and Ronitt Rubinfeld. Improved Massively Parallel Computation Algorithms for MIS, Matching, and Vertex Cover. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 129–138, 2018. doi:10.1145/3212734.3212743.
- 23 Christian Glazik, Jan Schiemann, and Anand Srivastav. Finding Euler Tours in One Pass in the W-Streaming Model with $O(n \log(n))$ RAM. *CoRR*, abs/1710.04091, 2017. arXiv:1710.04091.
- 24 A. Goldberg, S. Plotkin, and G. Shannon. Parallel Symmetry-breaking in Sparse Graphs. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, STOC '87*, pages 315–324, New York, NY, USA, 1987. ACM. doi:10.1145/28395.28429.
- 25 Andrew V. Goldberg and Serge A. Plotkin. Parallel $(\Delta + 1)$ -coloring of Constant-degree Graphs. *Inf. Process. Lett.*, 25(4):241–245, June 1987. doi:10.1016/0020-0190(87)90169-4.
- 26 Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, Searching, and Simulation in the MapReduce Framework. In *Algorithms and Computation - 22nd International Symposium, ISAAC 2011, Yokohama, Japan, December 5-8, 2011. Proceedings*, pages 374–383, 2011. doi:10.1007/978-3-642-25591-5_39.

- 27 David G Harris, Johannes Schneider, and Hsin-Hao Su. Distributed $(\Delta + 1)$ -coloring in sublogarithmic rounds. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 465–478. ACM, 2016.
- 28 Nicholas J. A. Harvey, Christopher Liaw, and Paul Liu. Greedy and Local Ratio Algorithms in the MapReduce Model. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA '18*, pages 43–52, New York, NY, USA, 2018. ACM. doi:10.1145/3210377.3210386.
- 29 Öjvind Johansson. Simple Distributed $\Delta+1$ -coloring of Graphs. *Inf. Process. Lett.*, 70(5):229–232, 1999. doi:10.1016/S0020-0190(99)00064-2.
- 30 Tomasz Jurdzinski and Krzysztof Nowicki. MST in $O(1)$ rounds of congested clique. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2620–2632, 2018. doi:10.1137/1.9781611975031.167.
- 31 Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A Model of Computation for MapReduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 938–948, 2010. doi:10.1137/1.9781611973075.76.
- 32 Fabian Kuhn and Rogert Wattenhofer. On the Complexity of Distributed Graph Coloring. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing, PODC '06*, pages 7–15, New York, NY, USA, 2006. ACM. doi:10.1145/1146381.1146387.
- 33 Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in MapReduce. In *SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Jose, CA, USA, June 4-6, 2011 (Co-located with FCRC 2011)*, pages 85–94, 2011. doi:10.1145/1989493.1989505.
- 34 Nathan Linial. Locality in Distributed Graph Algorithms. *SIAM J. Comput.*, 21(1):193–201, February 1992. doi:10.1137/0221015.
- 35 M Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, STOC '85*, pages 1–10, New York, NY, USA, 1985. ACM. doi:10.1145/22145.22146.
- 36 Alessandro Panconesi and Aravind Srinivasan. Improved Distributed Algorithms for Coloring and Network Decomposition Problems. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing, STOC '92*, pages 581–592, New York, NY, USA, 1992. ACM. doi:10.1145/129712.129769.
- 37 Merav Parter. $(\Delta + 1)$ Coloring in the Congested Clique Model. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 160:1–160:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.160.
- 38 Merav Parter and Hsin-Hao Su. Randomized $(\Delta + 1)$ -Coloring in $O(\log^* \Delta)$ Congested Clique Rounds. In *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, pages 39:1–39:18, 2018. doi:10.4230/LIPIcs.DISC.2018.39.
- 39 Vadim G Vizing. On an estimate of the chromatic class of a p-graph. *Discret Analiz*, 3:25–30, 1964.

Quantum Algorithms for Classical Probability Distributions

Aleksandrs Belovs

Faculty of Computing, University of Latvia, Riga, Latvia
aleksandrs.belovs@lu.lv

Abstract

We study quantum algorithms working on classical probability distributions. We formulate four different models for accessing a classical probability distribution on a quantum computer, which are derived from previous work on the topic, and study their mutual relationships.

Additionally, we prove that quantum query complexity of distinguishing two probability distributions is given by their inverse Hellinger distance, which gives a quadratic improvement over classical query complexity for any pair of distributions.

The results are obtained by using the adversary method for state-generating input oracles and for distinguishing probability distributions on input strings.

2012 ACM Subject Classification Theory of computation → Quantum query complexity

Keywords and phrases quantum query complexity, quantum adversary method, distinguishing probability distributions, Hellinger distance

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.16

Related Version A full version of the paper is available at <https://arxiv.org/abs/1904.02192>.

Funding This research is partly supported by the ERDF grant number 1.1.1.2/VIAA/1/16/113.

Acknowledgements Most of all I would like to thank Anras Gilyen for the suggestion to work on this problem. I am also grateful to Frederic Magniez, Shalev Ben-David, and Anurag Anshu for useful discussions, as well as to anonymous reviewers for their comments. Part of this research was performed while at the Institute for Quantum Computing in Waterloo, Canada. I would like to thank Ashwin Nayak for hospitality.

1 Introduction

It is customary for a quantum algorithm to receive its input and produce its output in the form of a classical string of symbols, quantized in the form of an oracle. This is purely classical way to store information, and, given intrinsic quantum nature of quantum algorithms, this might be not the best interface for many tasks. Moreover, even *classical* algorithms make use of other interfaces as well. For instance, classical algorithms can receive and produce *samples* from some probability distribution. In this paper we study quantum algorithms working with classical probability distributions.

1.1 Models

We analyse previously used models of accessing classical probability distributions by quantum algorithms. We prove and conjecture some relations between them. We give more detail in Section 3, but for now let us very briefly introduce the models.

In one of the models, used in, e.g., [16, 18, 27, 25], the probability distribution is encoded as a frequency of a symbol in a given input string, which the quantum algorithm accesses via the standard input oracle. In another model, e.g., [17, 2, 6], the input probability distribution



© Aleksandrs Belovs;

licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 16; pp. 16:1–16:11

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is given through a quantum oracle that prepares a state in the form $\sum_a \sqrt{p_a}|a\rangle$. Finally, one more model, used in [26, 20, 19], is similar but with additional state tensored with each $|a\rangle$.

This is the latter model that we champion in this paper. We find this model particularly relevant because of our belief that an input oracle should be easily interchangeable with a quantum subroutine, see discussion in [26]. It is relatively easy to see what it means for a quantum algorithm to output a probability distribution: just measure one of the registers of its final state. The latter model precisely encompasses all such subroutines. We conjecture that this model is equivalent to the first model, see also [19], where a similar conjecture is made.

1.2 Distinguishing two Probability Distributions

Additionally, we study the problem of distinguishing two probability distributions. This might be the most fundamental problem one can formulate in these settings. Given two fixed probability distributions p and q , and given an input oracle encoding one of them, the task is to detect which one, p or q , the oracle encodes. To the best of our knowledge, this particular problem has not been studied in quantum settings, although similar problems of testing the distance between two distributions [16] and testing whether the input distribution is equal to some fixed distribution [18] have been already studied.

Classically one needs $\Theta(1/d_H(p, q)^2)$ samples to solve this problem for any p and q , where d_H stands for Hellinger distance. This result is considered “folklore”, see, e.g. [7, Chapter 4]. We prove that for any p and q and for any of the models of access described above, query complexity of this problem is $\Theta(1/d_H(p, q))$. This constitutes quadratic improvement over classical algorithm for *any* pair of distributions p and q . Moreover, our algorithm also admits a simple low-level implementation, which is efficient assuming the distributions p and q can be efficiently processed.

1.3 Techniques

Our main technical tool for proving the upper bound is the version of the adversary bound for state-generating oracles, which is a special case of the adversary bound for general input oracles [11]. It is stated in the form of a relative γ_2 -norm and generalises the dual formulation of the general adversary bound [28, 29] for function evaluation, as well as for other problems [5, 24]. The dual adversary bound has been used rather successfully in construction of quantum algorithms, as in terms of span programs and learning graphs [9, 23, 13, 8, 22], as in an unrelated fashion [10, 4]. Our work gives yet another application of these techniques for construction of quantum algorithms.

Our upper bound naturally follows from the analysis of the γ_2 -norm optimisation problem associated with the task. We also compare our techniques with more standard ones involving quantum rejection sampling and amplitude amplification in the spirit of [20] and show that our techniques give a slightly better result.

As for the lower bound, we make use of the version of the adversary bound from [12]. This is a simple generalisation of the primal version of the general adversary bound [21] for function evaluation, and it is tailored for the task we are interested in: distinguishing two probability distributions on input strings. Our lower bound is surprisingly simple and gives a very intuitive justification of the significance of Hellinger distance for this problem.

2 Preliminaries

We mostly use standard linear-algebraic notation. We use ket-notation for vectors representing quantum states, but generally avoid it. We use A^* to denote conjugate operators (transposed and complex-conjugated matrices). For P a predicate, we use 1_P to denote 1 if P is true, and 0 if P is false. We use $[n]$ to denote the set $\{1, 2, \dots, n\}$.

It is unfortunate that the same piece of notation, \oplus , is used both for direct sum of matrices and direct sum of vectors, which is in conflict with each other if a vector, as it often does, gets interpreted as a column-matrix. Since we will extensively use both these operations in this paper, let us agree that \boxplus denotes direct sum of vectors, and \oplus always denotes direct sum of matrices. Thus, in particular, for $u, v \in \mathbb{R}^m$, we have

$$u \boxplus v = \begin{pmatrix} u_1 \\ \vdots \\ u_m \\ v_1 \\ \vdots \\ v_m \end{pmatrix} \quad \text{and} \quad u \oplus v = \begin{pmatrix} u_1 & 0 \\ \vdots & \vdots \\ u_m & 0 \\ 0 & v_1 \\ \vdots & \vdots \\ 0 & v_m \end{pmatrix}.$$

We often treat scalars as 1×1 -matrices which may be also thought as vectors.

2.1 Relative γ_2 -norm

In this section, we state the relative γ_2 -norm and formulate some of its basic properties. All the results are from [11].

► **Definition 1** (Relative γ_2 -norm). *Let $\mathcal{X}_1, \mathcal{X}_2, \mathcal{Z}_1$ and \mathcal{Z}_2 be vector spaces, and D_1 and D_2 be some sets of labels. Let $A = \{A_{xy}\}$ and $\Delta = \{\Delta_{xy}\}$, where $x \in D_1$ and $y \in D_2$, be two families of linear operators: $A_{xy}: \mathcal{Z}_2 \rightarrow \mathcal{Z}_1$ and $\Delta_{xy}: \mathcal{X}_2 \rightarrow \mathcal{X}_1$. The relative γ_2 -norm,*

$$\gamma_2(A|\Delta) = \gamma_2(A_{xy} \mid \Delta_{xy})_{x \in D_1, y \in D_2},$$

is defined as the optimal value of the following optimisation problem, where Υ_x and Φ_y are linear operators,

$$\text{minimise} \quad \max \left\{ \max_{x \in D_1} \|\Upsilon_x\|^2, \max_{y \in D_2} \|\Phi_y\|^2 \right\} \quad (1a)$$

$$\text{subject to} \quad A_{xy} = \Upsilon_x^*(\Delta_{xy} \otimes I_{\mathcal{W}})\Phi_y \quad \text{for all } x \in D_1 \text{ and } y \in D_2; \quad (1b)$$

$$\mathcal{W} \text{ is a vector space, } \Upsilon_x: \mathcal{Z}_1 \rightarrow \mathcal{X}_1 \otimes \mathcal{W}, \quad \Phi_y: \mathcal{Z}_2 \rightarrow \mathcal{X}_2 \otimes \mathcal{W}. \quad (1c)$$

This is a generalisation of the usual γ_2 -norm, also known as Schur (Hadamard) product operator norm [14].

In a quantum algorithm with general input oracles, the input oracle performs some unitary operation O on some fixed Hilbert space. The algorithm can execute either O or its inverse O^{-1} on some register. Each execution counts as one query. It is known that O is equal to one O_x out of a set of possible input unitaries, where x ranges over some set D of labels. If $O = O_x$, the algorithm has to perform a unitary V_x on some specified part of its work-space. The algorithm knows in advance all possible O_x and which V_x corresponds to each O_x , but it does not know which O_x it is given in a specific execution. The adversary bound corresponding to this problem is $\gamma_2(V_x - V_y \mid O_x - O_y)_{x, y \in D}$. This bound is *semi-tight*: it is a lower bound on the exact version of the problem and an upper bound on the approximate version.

16:4 Quantum Algorithms for Classical Probability Distributions

The γ_2 -norm formalism is modular in the sense that the general task of implementing a unitary can be replaced by something more specific. For instance, assume that our task is to evaluate a function $f(x)$. Then the adversary bound reads as $\gamma_2(1_{f(x) \neq f(y)} \mid O_x - O_y)_{x,y \in D}$. In this case, the bound is tight: it is also a lower bound on the approximate version of the problem.

As another example, consider the standard input oracle O_x encoding a string $x \in [q]^n$. It works as $O_x: |i\rangle|0\rangle \mapsto |i\rangle|x_i\rangle$, which can be seen as a direct sum of oracles performing transformation $|0\rangle \mapsto |x_i\rangle$. Using the modular approach, the corresponding adversary bound becomes $\gamma_2(1_{f(x) \neq f(y)} \mid \bigoplus_j 1_{x_j \neq y_j})_{x,y \in D}$, where \bigoplus stands for direct sum of matrices (resulting in a diagonal matrix). This is equivalent to the usual version of dual adversary for function evaluation (up to a constant factor).

Now we consider state-generating input oracles¹. In this case, the input to the algorithm is given by a state $\psi \in \mathbb{C}^m$, and the algorithm should work equally well for any unitary performing the transformation $O: |0\rangle \mapsto |\psi\rangle$. Without loss of generality, we may assume that $e_0 = |0\rangle$ is orthogonal to \mathbb{C}^m , thus the operator O above works in \mathbb{C}^{m+1} .

The corresponding γ_2 -object can be defined in two alternative ways:

$$L_\psi = \psi e_0^* + e_0 \psi^* \quad \text{or} \quad L_\psi = \psi \oplus \psi^*.$$

In the second expression, ψ is an $m \times 1$ -matrix and ψ^* is a $1 \times m$ -matrix, the resulting matrix being of size $(m+1) \times (m+1)$. In the case of a function-evaluation problem, the corresponding adversary bound is $\gamma_2(1_{f(x) \neq f(y)} \mid L_{\psi_x} - L_{\psi_y})_{x,y \in D}$.

Let us also state the version of the adversary bound for the decision problem with state-generating input oracles. This is the version we will use further in the paper. Assume we have a collection of states $\psi_x \in \mathcal{X}$ for $x \in D_0$, and a collection of states $\psi_y \in \mathcal{X}$ for $y \in D_1$. The task is to distinguish the two classes of states. Let $D = D_0 \cup D_1$. Using the general case, we obtain the following version of the adversary bound.

► **Theorem 2.** *The quantum query complexity of the decision problem with state-generating oracles as above is equal to $\gamma_2(1 \mid L_{\psi_x} - L_{\psi_y})_{x \in D_0, y \in D_1}$ up to a constant factor.*

An explicit optimisation problem for $\gamma_2(1 \mid L_{\psi_x} - L_{\psi_y})_{x \in D_0, y \in D_1}$ is given by

$$\begin{aligned} \text{minimise} \quad & \max_{z \in D} (\|u_z\|^2 + \|v_z\|^2) \\ \text{subject to} \quad & \langle v_x, (\psi_x - \psi_y) \otimes u_y \rangle + \langle (\psi_x - \psi_y) \otimes u_x, v_y \rangle = 1 \quad \forall x \in D_0, y \in D_1; \\ & u_z \in \mathcal{W}, \quad v_z \in \mathcal{X} \otimes \mathcal{W} \quad \forall z \in D. \end{aligned} \quad (2)$$

This result follows from general results of [11], see the full version of the paper, where we also give a stand-alone implementation and analysis of the corresponding quantum algorithm.

3 Models

In this section we formally define four different models how a quantum algorithm can access a classical probability distribution $p = (p_a)_{a \in A}$. These models were briefly explained in the introduction. We would like to understand relations between them, and, ideally, prove some equivalences between them.

¹ The results below will appear in an updated version of [11] (to appear). Alternatively, refer to the full version of the paper.

- (i) A standard input oracle encoding a string $x \in A^n$ for some relatively large n , where p_a is given as the frequency of a in x :

$$p_a = \frac{1}{n} \left| \{i \mid x_i = a\} \right|.$$

- (ii) A standard input oracle encoding a string $x \in A^n$ for some relatively large n , where each x_i is drawn independently at random from p .
 (iii) A quantum procedure that generates the state

$$\mu_p = \sum_a \sqrt{p_a} |a\rangle = \bigoplus_a \sqrt{p_a}. \quad (3)$$

- (iv) A quantum procedure that generates a state of the form

$$\sum_a \sqrt{p_a} |a\rangle |\psi_a\rangle = \bigoplus_a \sqrt{p_a} \psi_a, \quad (4)$$

where ψ_a are arbitrary unit vectors.

As mentioned in the introduction, model (i) is used in [16, 18, 27, 25]. It has a downside that the probabilities p_a must be multiples of $1/n$. All other models are free from this assumption.

Model (ii) seems like the most obvious way to encode probability distribution as a classical string, which a quantum algorithm can later gain access to. Up to our knowledge, this model has not been previously used. It has a downside that the distribution p is encoded as a probability distribution over possible input strings, which is not usual for quantum algorithms. The acceptance probability of the quantum algorithm depends both on the randomness introduced by the algorithm and the randomness in the input.

Model (iii) is the one used in [17, 2, 6]. And model (iv) is used in [26, 20, 19]. Both of these two models assume that the input oracle prepares a quantum state, which again is not very common for quantum algorithms.

► **Proposition 3.** *We have the following relations between these models.*

- (a) *Models (i) and (ii) are equivalent assuming n is large enough. More precisely, no quantum algorithm can distinguish models (i) and (ii) encoding the same probability distribution unless it makes $\Omega(n^{1/3})$ queries.*
 (b) *Model (iv) is more general than model (i). This means that any algorithm working in model (iv) can be turned into an algorithm working in model (i) with the same query complexity.*
 (c) *Model (iv) is strictly more general than model (iii). This means there exist problems where model (iii) allows substantially smaller query complexity than model (iv).*

Proof. We leave (a) for the end of the proof, and let us start with (b). Note that using one query to the input oracle of model (i), it is possible to prepare that state

$$\frac{1}{\sqrt{n}} \sum_i |i\rangle |x_i\rangle = \sum_{a \in A} \left[\frac{1}{\sqrt{n}} \sum_{i: x_i = a} |i\rangle \right] \otimes |a\rangle,$$

which is a legitimate input state in model (iv) if one swaps the registers.

Now let us prove (c). It is obvious that model (iv) is more general than model (iii). To prove that (iii) cannot simulate (iv), consider the collision problem [15]. In this problem, a function $f: [n] \rightarrow [n]$ is given, and one has to distinguish whether f is 1-to-1 or 2-to-1. In terms of model (i), this boils down to distinguishing a probability distribution p which is uniform on $[n]$ from a probability distribution q which is uniform on half of $[n]$.

16:6 Quantum Algorithms for Classical Probability Distributions

In model (iii), this problem can be solved in $O(1)$ queries because the state μ_p as in (3) has inner product $1/\sqrt{2}$ with all μ_q . On the other hand, by [1, 3], quantum query complexity of this problem in model (i) is $\Omega(n^{1/3})$. As model (iv) is more general than model (i), this gives the required lower bound.

To prove (a), we show that if one can distinguish models (i) and (ii), one can distinguish a random function from a random permutation, and the result follows from the lower bound of $\Omega(n^{1/3})$ for this task from [30]. Indeed, let p be a probability distribution and let y be a fixed string encoding p as in model (i). Let $\sigma: [n] \rightarrow [n]$ be a function, and consider the input string x given by $x_i = y_{\sigma(i)}$, which can be simulated given oracle access to σ (as the string y is fixed). If σ is a random permutation, then x is a uniformly random input string from model (i). If σ is a random function, then x is distributed as in model (ii). ◀

4 Distinguishing Two Probability Distributions

Recall the definition of Hellinger distance between two probability distributions p and q on the same space A :

$$d_H(p, q) = \sqrt{\frac{1}{2} \sum_{a \in A} (\sqrt{p_a} - \sqrt{q_a})^2}.$$

Up to a constant factor, it equals $\|\mu_p - \mu_q\|$ and $1 - \langle \mu_p, \mu_q \rangle$, where μ_p and μ_q are as in (3).

In this section, we prove the following result:

► **Theorem 4.** *For any two probability distributions p and q on the same space A , and any model of accessing them from Section 3, the quantum query complexity of distinguishing p and q is*

$$\Theta\left(\frac{1}{d_H(p, q)}\right).$$

Note that this is quadratically better than complexity of the best classical algorithm for every choice of p and q . Note also that for this problem model (iii) is equal in strength to the remaining models.

The proof of main involves proving lower and upper bounds in all four models, but, luckily, we can use relations from Proposition 3. The outline of the proof is as follows. We prove upper bound in model (iv), which implies upper bounds in all other models as model (iv) is the most general of them. As for the lower bounds, we prove it for model (ii), which implies lower bounds in models (i) and (iv). For model (iii), we prove the lower bound independently. As a bonus, we prove an upper bound in model (iii) as a warm-up for the upper bound in model (iv).

In most of the proofs, we will use α for the angle between the vectors μ_p and μ_q . Note that

$$\alpha = \Theta(\|\mu_p - \mu_q\|) = \Theta(d_H(p, q)).$$

4.1 Analysis in Model (iii)

In this section, we analyse the problem in model (iii).

▷ **Claim 5.** Quantum query complexity of distinguishing probability distributions p and q in model (iii) is $\Theta(1/d_H(p, q))$.

Proof. Let us start with the upper bound. Let O be the input oracle, and let U be a unitary that maps $|\mu_p\rangle$ into $|0\rangle$ and $|\mu_q\rangle$ into $\cos\alpha|0\rangle + \sin\alpha|1\rangle$. Now use quantum amplitude amplification on the unitary UO amplifying for the value $|1\rangle$. The algorithm can be also made exact using exact quantum amplitude amplification.

Now let us prove the lower bound. Let O_p be the input oracle exchanging $|0\rangle$ and $|\mu_p\rangle$ and leaving the vectors orthogonal to them intact. Similarly, let O_q exchange $|0\rangle$ and $|\mu_q\rangle$. Simple linear algebra shows $\|O_p - O_q\| = O(\alpha)$. Let \mathcal{A}^O be a query algorithm making t queries to O and distinguishing O_p from O_q . Then,

$$\|\mathcal{A}^{O_p} - \mathcal{A}^{O_q}\| \leq t\|O_p - O_q\| = O(t\alpha).$$

As this must be $\Omega(1)$, we get that $t = \Omega(1/\alpha)$. ◁

4.2 Upper Bound in Model (iv)

The aim of this section is to prove the following claim.

▷ **Claim 6.** Quantum query complexity of distinguishing probability distributions p and q in model (iv) is $O(1/d_H(p, q))$.

We prove this claim by constructing a feasible solution to (2). In the full version of the paper, we explain how to implement this algorithm time-efficiently and give a comparison to an algorithm using more typical techniques.

Let ψ and ϕ be some vectors encoding p and q , respectively, as in model (iv). That is,

$$\psi = \bigoplus_a \sqrt{p_a} \psi_a, \quad \text{and} \quad \phi = \bigoplus_a \sqrt{q_a} \phi_a,$$

where ψ_a and ϕ_a are some normalised vectors. Our goal is to come up with a feasible solution to (2) with ψ_x and ψ_y replaced by ψ and ϕ .

We first analyse a pair of vectors $\sqrt{p_a} \psi_a$ and $\sqrt{q_a} \phi_a$ for a fixed a . We would like to get a construction in the spirit of (2) that “erases” directions ψ_a and ϕ_a , and only depends on the norms $\sqrt{p_a}$ and $\sqrt{q_a}$. One way is to use the following identity:

$$\left\langle \sqrt{p_a} \psi_a, \sqrt{p_a} \psi_a - \sqrt{q_a} \phi_a \right\rangle + \left\langle \sqrt{p_a} \psi_a - \sqrt{q_a} \phi_a, \sqrt{q_a} \phi_a \right\rangle = p_a - q_a. \quad (5)$$

We combine this identity over all a , add weights c_a , and re-normalise:

$$\begin{aligned} & \left\langle \frac{\bigoplus_a c_a \sqrt{p_a} \psi_a}{\sqrt[4]{\sum_a c_a^2 p_a}}, (\psi - \phi) \cdot \sqrt[4]{\sum_a c_a^2 p_a} \right\rangle \\ & + \left\langle (\psi - \phi) \cdot \sqrt[4]{\sum_a c_a^2 q_a}, \frac{\bigoplus_a c_a \sqrt{q_a} \phi_a}{\sqrt[4]{\sum_a c_a^2 q_a}} \right\rangle = \sum_a c_a (p_a - q_a), \end{aligned}$$

which gives

$$\gamma_2 \left(\sum_a c_a (p_a - q_a) \mid L_\psi - L_\phi \right)_{\psi, \phi} \leq \sqrt{\sum_a c_a^2 p_a} + \sqrt{\sum_a c_a^2 q_a}.$$

16:8 Quantum Algorithms for Classical Probability Distributions

Dividing by $\sum_a c_a(p_a - q_a)$, we get that complexity of distinguishing p from q is at most

$$O\left(\frac{\sqrt{\sum_a c_a^2 p_a} + \sqrt{\sum_a c_a^2 q_a}}{\sum_a c_a(p_a - q_a)}\right). \quad (6)$$

Using triangle inequality

$$\sqrt{\sum_a c_a^2 (\sqrt{p_a} + \sqrt{q_a})^2} \leq \sqrt{\sum_a c_a^2 p_a} + \sqrt{\sum_a c_a^2 q_a} \leq 2\sqrt{\sum_a c_a^2 (\sqrt{p_a} + \sqrt{q_a})^2},$$

so (6) is equivalent to

$$O\left(\frac{\sqrt{\sum_a c_a^2 (\sqrt{p_a} + \sqrt{q_a})^2}}{\sum_a c_a(p_a - q_a)}\right).$$

Now it is easy to see that it is minimised to

$$O\left(\frac{1}{\sqrt{\sum_a (\sqrt{p_a} - \sqrt{q_a})^2}}\right) = O\left(\frac{1}{d_H(p, q)}\right)$$

when $c_a = (\sqrt{p_a} - \sqrt{q_a})/(\sqrt{p_a} + \sqrt{q_a})$.

4.3 Lower Bound in Model (ii)

We use the following version of the adversary lower bound from [12].

► **Theorem 7.** *Assume \mathcal{A} is a quantum algorithm that makes T queries to the input string $x = (x_1, \dots, x_n) \in D$, with $D = A^n$, and then either accepts or rejects. Let P and Q be two probability distributions on D , and p_x and q_y denote probabilities of x and y in P and Q , respectively. Let s_P and s_Q be acceptance probability of \mathcal{A} when x is sampled from P and Q , respectively. Then,*

$$T = \Omega\left(\min_{j \in [n]} \frac{\delta_P^* \Gamma \delta_Q - \tau(s_P, s_Q) \|\Gamma\|}{\|\Gamma \circ \Delta_j\|}\right), \quad (7)$$

for any $D \times D$ matrix Γ with real entries. Here, $\delta_P[x] = \sqrt{p_x}$ and $\delta_Q[y] = \sqrt{q_y}$ are unit vectors in \mathbb{R}^D ; for $j \in [n]$, the $D \times D$ matrix Δ_j is defined by $\Delta_j[x, y] = 1_{x_j \neq y_j}$; and

$$\tau(s_P, s_Q) = \sqrt{s_P s_Q} + \sqrt{(1 - s_P)(1 - s_Q)} \leq 1 - \frac{|s_P - s_Q|^2}{8}. \quad (8)$$

In our case, $\delta_P = \mu_p^{\otimes n}$ and $\delta_Q = \mu_q^{\otimes n}$. We construct Γ as a tensor power $G^{\otimes n}$, where G is an $A \times A$ matrix satisfying

$$G\mu_q = \mu_p, \quad \|G\| = 1, \quad \text{and} \quad \|G \circ \Delta\| \text{ is as small as possible,}$$

where Δ is the $A \times A$ matrix given by $A[a, b] = 1_{a \neq b}$. Then,

$$\delta_P^* \Gamma \delta_Q = \|\Gamma\| = 1, \quad \text{and} \quad \|\Gamma \circ \Delta_j\| = \|G \circ \Delta\|,$$

and adv gives the lower bound of $\Omega(1/\|G \circ \Delta\|)$.

We construct G as follows. Recall that α is the angle between μ_q and μ_p . Then, G is rotation by the angle α in the plane spanned by μ_q and μ_p and homothety with coefficient $\cos \alpha$ on its orthogonal complement. That is, in an orthonormal basis where the first two vectors span the plane of μ_q and μ_p , we have

$$G = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 & \cdots & 0 \\ \sin \alpha & \cos \alpha & 0 & \cdots & 0 \\ 0 & 0 & \cos \alpha & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \cos \alpha \end{pmatrix}.$$

Clearly, $G\mu_q = \mu_p$ and $\|G\| = 1$. Let $G' = G - \cos \alpha I$. We have

$$\|G \circ \Delta\| = \|G' \circ \Delta\| \leq 2\|G'\| = 2\sin \alpha = O(d_H(p, q)).$$

For the inequality we used that $\gamma_2(\Delta) \leq 2$, see [24, Theorem 3.4]. This gives the required lower bound.

5 Summary and Future Work

In this paper we considered quantum algorithms dealing with classical probability distributions. We identified four different models, and proved various relations between them. We conjecture that models (i), (ii) and (iv) are equivalent.

Also, we considered the problem of distinguishing two probability distributions and obtained precise characterisation of its quantum query complexity in all four models in terms of Hellinger distance between the probability distributions. The complexity turned out to be exactly quadratically smaller than the classical complexity of this problem for all pairs of distributions.

We showed that the corresponding algorithm can be implemented efficiently given that the probability distributions p and q can be handled efficiently. We also compared our algorithm with a more standard approach using rejection sampling and amplitude estimation.

This raises a number of interesting open problems. The first one is to prove or disprove the conjecture that models (i) and (iv) are equivalent. Another interesting problem is to come up with a nice γ_2 -characterisation of probability distribution oracles like `gamma2StatePreparing` characterises state-generating oracles. Unfortunately, we do not have any hypothesis of how this characterisation might look like. Finally, we would be interested in further quantum algorithms based on techniques of Section 4.2.

References

- 1 Scott Aaronson and Yaoyun Shi. Quantum Lower Bounds for the Collision and the Element Distinctness Problems. *Journal of the ACM*, 51(4):595–605, 2004. doi:10.1145/1008731.1008735.
- 2 Dorit Aharonov and Amnon Ta-Shma. Adiabatic quantum state generation. *SIAM Journal on Computing*, 37(1):47–82, 2007. doi:doi.org/10.1137/060648829.
- 3 Andris Ambainis. Polynomial Degree and Lower Bounds in Quantum Complexity: Collision and Element Distinctness with Small Range. *Theory of Computing*, 1:37–46, 2005.
- 4 Andris Ambainis, Aleksandrs Belovs, Oded Regev, and Ronald de Wolf. Efficient Quantum Algorithms for (Gapped) Group Testing and Junta Testing. In *Proc. of 27th ACM-SIAM SODA*, pages 903–922, 2016. doi:10.1137/1.9781611974331.ch65.


- 5 Andris Ambainis, Loïck Magnin, Martin Rötteler, and Jérémie Roland. Symmetry-assisted adversaries for quantum state generation. In *Proc. of 26th IEEE CCC*, pages 167–177, 2011. doi:10.1109/CCC.2011.24.
- 6 Alp Atıcı and Rocco A. Servedio. Improved bounds on quantum learning algorithms. *Quantum Information Processing*, 4(5):355–386, 2005. doi:10.1007/s11128-005-0001-2.
- 7 Ziv Bar-Yossef. *The Complexity of Massive Data Set Computations*. PhD thesis, UC Berkeley, 2002.
- 8 Aleksandrs Belovs. Learning-graph-based Quantum Algorithm for k -distinctness. In *Proc. of 53rd IEEE FOCS*, pages 207–216, 2012. doi:10.1109/FOCS.2012.18.
- 9 Aleksandrs Belovs. Span programs for functions with constant-sized 1-certificates. In *Proc. of 44th ACM STOC*, pages 77–84, 2012. doi:10.1145/2213977.2213985.
- 10 Aleksandrs Belovs. Quantum Algorithms for Learning Symmetric Juntas via the Adversary Bound. *Computational Complexity*, 24(2):255–293, 2015. doi:10.1007/s00037-015-0099-2.
- 11 Aleksandrs Belovs. Variations on Quantum Adversary, 2015. arXiv:1504.06943.
- 12 Aleksandrs Belovs, Gilles Brassard, Peter Høyer, Marc Kaplan, Sophie Laplante, and Louis Salvail. Provably secure key establishment against quantum adversaries. In *Proc. of 12th TQC*, volume 73 of *LIPICs*, pages 3:1–3:17. Dagstuhl, 2018.
- 13 Aleksandrs Belovs and Ben W. Reichardt. Span programs and quantum algorithms for st -connectivity and claw detection. In *Proc. of 20th ESA*, volume 7501 of *LNCS*, pages 193–204. Springer, 2012. doi:10.1007/978-3-642-33090-2_18.
- 14 Rajendra Bhatia. *Positive definite matrices*. Princeton University Press, 2009.
- 15 Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum cryptanalysis of hash and claw-free functions. In *Proc. of 3rd LATIN*, volume 1380 of *LNCS*, pages 163–169. Springer, 1998. doi:10.1007/BFb0054319.
- 16 Sergey Bravyi, Aram W. Harrow, and Avinatan Hassidim. Quantum algorithms for testing properties of distributions. *IEEE Transactions on Information Theory*, 57:3971–3981, 2011. doi:10.1109/TIT.2011.2134250.
- 17 Nader H. Bshouty and Jeffrey C. Jackson. Learning DNF over the uniform distribution using a quantum example oracle. *SIAM Journal on Computing*, 28(3):1136–1153, 1998. doi:10.1137/S0097539795293123.
- 18 Sourav Chakraborty, Eldar Fischer, Arie Matsliah, and Ronald de Wolf. New Results on Quantum Property Testing. In *Proc. of 30th FSTTCS*, volume 8 of *LIPICs*, pages 145–156. Dagstuhl, 2010. doi:10.4230/LIPICs.FSTTCS.2010.145.
- 19 András Gilyén and Tongyang Li. Distributional property testing in a quantum world, 2019. arXiv:1902.00814.
- 20 Yassine Hamoudi and Frédéric Magniez. Quantum Chebyshev’s Inequality and Applications, 2018. arXiv:1807.06456.
- 21 Peter Høyer, Troy Lee, and Robert Špalek. Negative weights make adversaries stronger. In *Proc. of 39th ACM STOC*, pages 526–535, 2007. doi:10.1145/1250790.1250867.
- 22 Michael Jarret, Stacey Jeffery, Shelby Kimmel, and Alvaro Piedrafita. Quantum Algorithms for Connectivity and Related Problems. In *Proc. of 26th ESA*, volume 112 of *LIPICs*, pages 49:1–49:13. Dagstuhl, 2018. doi:10.4230/LIPICs.ESA.2018.49.
- 23 Troy Lee, Frédéric Magniez, and Miklos Santha. A learning graph based quantum query algorithm for finding constant-size subgraphs. *Chicago Journal of Theoretical Computer Science*, 2012(10), 2012.
- 24 Troy Lee, Rajat Mittal, Ben W. Reichardt, Robert Špalek, and Mario Szegedy. Quantum query complexity of state conversion. In *Proc. of 52nd IEEE FOCS*, pages 344–353, 2011. doi:10.1109/FOCS.2011.75.
- 25 Tongyang Li and Xiaodi Wu. Quantum query complexity of entropy estimation. *IEEE Transactions on Information Theory*, page 1–1, 2018. doi:10.1109/TIT.2018.2883306.
- 26 Ashley Montanaro. Quantum speedup of Monte Carlo methods. *Proceedings of the Royal Society A*, 471(2181), 2015. doi:10.1098/rspa.2015.0301.

- 27 Ashley Montanaro. The quantum complexity of approximating the frequency moments. *Quantum Information & Computation*, 16:1169–1190, 2016.
- 28 Ben W. Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every Boolean function. In *Proc. of 50th IEEE FOCS*, pages 544–551, 2009. doi:10.1109/FOCS.2009.55.
- 29 Ben W. Reichardt. Reflections for quantum query algorithms. In *Proc. of 22nd ACM-SIAM SODA*, pages 560–569, 2011. doi:10.1137/1.9781611973082.44.
- 30 Mark Zhandry. A Note on the Quantum Collision and Set Equality Problems. *Quantum Information & Computation*, 15(7&8):557–567, 2015.

More Applications of the d -Neighbor Equivalence: Connectivity and Acyclicity Constraints

Benjamin Bergounoux¹ 

Université Paris Diderot, IRIF, CNRS, Paris, France
bergounoux@irif.fr

Mamadou Moustapha Kanté 

Université Clermont Auvergne, LIMOS, CNRS, Aubière, France
mamadou.kante@uca.fr

Abstract

In this paper, we design a framework to obtain efficient algorithms for several problems with a global constraint (acyclicity or connectivity) such as CONNECTED DOMINATING SET, NODE WEIGHTED STEINER TREE, MAXIMUM INDUCED TREE, LONGEST INDUCED PATH, and FEEDBACK VERTEX SET. For all these problems, we obtain $2^{O(k)} \cdot n^{O(1)}$, $2^{O(k \log(k))} \cdot n^{O(1)}$, $2^{O(k^2)} \cdot n^{O(1)}$ and $n^{O(k)}$ time algorithms parameterized respectively by clique-width, \mathbb{Q} -rank-width, rank-width and maximum induced matching width. Our approach simplifies and unifies the known algorithms for each of the parameters and match asymptotically also the running time of the best algorithms for basic NP-hard problems such as VERTEX COVER and DOMINATING SET. Our framework is based on the d -neighbor equivalence defined in [Bui-Xuan, Telle and Vatshelle, TCS 2013]. The results we obtain highlight the importance and the generalizing power of this equivalence relation on width measures. We also prove that this equivalence relation could be useful for MAX CUT: a W[1]-hard problem parameterized by clique-width. For this latter problem, we obtain $n^{O(k)}$, $n^{O(k)}$ and $n^{2^{O(k)}}$ time algorithm parameterized by clique-width, \mathbb{Q} -rank-width and rank-width.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms

Keywords and phrases connectivity problem, feedback vertex set, d -neighbor equivalence, σ, ρ -domination, clique-width, rank-width, mim-width

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.17

Related Version A full version of the paper is available at <https://arxiv.org/abs/1805.11275>.

Acknowledgements This work is supported by French Agency for Research under the GraphEN project (ANR-15-CE40-0009).

1 Introduction

Tree-width is one of the most well-studied graph parameters in the graph algorithm community, due to its numerous structural and algorithmic properties. Nevertheless, despite the broad interest on tree-width, only sparse graphs can have bounded tree-width. But, many NP-hard problems are tractable on dense graph classes. For many graph classes, this tractability can be explained through other width measures. The most remarkable ones are certainly clique-width [8], rank-width [18], and maximum induced matching width (*a.k.a.* mim-width) [23]. We obtain most of these parameters through the notion of *rooted layout* (see Section 2).

These other width measures have a modeling power strictly stronger than the modeling power of tree-width. For example, if a graph class has bounded tree-width, then it has bounded clique-width [8], but the converse is false as cliques have clique-width at most 2

¹ Corresponding author



and unbounded tree-width. While rank-width has the same modeling power as clique-width, mim-width has the strongest one among all these width measures and is even bounded on interval graphs [1]. Despite their generality, a lot of NP-hard problems admit polynomial time algorithms when one of these width measures is fixed. But, dealing with these width measures is known to be harder than manipulating tree-width.

Concerning their computations, it is not known whether the clique-width (respectively mim-width) of a graph can be approximated within a constant factor in time $f(k) \cdot n^{O(1)}$ (resp. $n^{f(k)}$) for some function f . However, for rank-width and its variant \mathbb{Q} -rank-width [19], there are efficient FPT algorithms for computing a decomposition approximating the width of the input graph [19, 20].

Finding efficient algorithms parameterized by one of these width measures is by now standard for problems based on local constraints [7, 22]. In contrast, the task is quite complicated for problems involving a global constraint, *e.g.*, connectivity or acyclicity. For a long time, our knowledge on the parameterized complexity of this latter kind of problems, with parameters the common width measures, was quite limited even for tree-width. For a while, the FPT community used to think that for problems involving global constraints the naive $k^{O(k)} \cdot n^{O(1)}$ time algorithm, k being the tree-width of the input graph, could not be improved. But, quite surprisingly, in 2011, Cygan et al. introduced in [9] a technique called *Cut & Count* to design Monte Carlo $2^{O(k)} \cdot n^{O(1)}$ time algorithms for a wide range of problems with global constraints, including HAMILTONIAN CYCLE, FEEDBACK VERTEX SET, and CONNECTED DOMINATING SET. Later, Bodlaender et al. proposed in [5] a general toolkit, called *rank-based approach*, to design deterministic $2^{O(k)} \cdot n$ time algorithms to solve a wider range of problems.

Recently, Bergougnoux and Kanté [3] adapted the rank-based approach of [5] to obtain $2^{O(k)} \cdot n$ time algorithms, k being the clique-width of a given decomposition, for many problems with a global constraint, *e.g.* CONNECTED DOMINATING SET and FEEDBACK VERTEX SET. Unlike tree-width and clique-width, algorithms parameterized by rank-width and mim-width for problems with a global constraint, were not investigated, except for some special cases such as FEEDBACK VERTEX SET [12, 15] and LONGEST INDUCED PATH [14].

One successful way to design efficient algorithms with these width measures is through the notion of *d-neighbor equivalence*. This concept was introduced by Bui-Xuan, Telle and Vatshelle in [7]. Formally, given $A \subseteq V(G)$ and $d \in \mathbb{N}^+$, two sets $X, Y \subseteq A$ are *d-neighbor equivalent w.r.t. A* if, for all $v \in V(G) \setminus A$, we have $\min(d, |N(v) \cap X|) = \min(d, |N(v) \cap Y|)$, where $N(v)$ is the set of neighbors of v in G . Notice that X and Y are 1-neighbor equivalent *w.r.t. A* if and only if both have the same neighborhood in $V(G) \setminus A$. The *d-neighbor equivalence* gives rise to a width measure, called in this paper *d-neighbor-width* (defined in Section 2). It is worth noticing that the boolean-width of a layout introduced in [6] corresponds to the binary logarithm of the 1-neighbor-width.

These notions were used by Bui-Xuan et al. in [7] to design efficient algorithms for the family of problems called (σ, ρ) -DOMINATING SET problems. This family of problems was introduced by Telle and Proskurowski in [22] (we define this family in Section 4). Many NP-hard problems based on local constraints belong to this family, see [7, Table 1].

Bui-Xuan et al. [7] designed an algorithm that, given a rooted layout \mathcal{L} , solve any (σ, ρ) -DOMINATING SET problem in time $\text{s-nec}_d(\mathcal{L})^{O(1)} \cdot n^{O(1)}$ where d is a constant depending on the considered problem. The known upper bounds on $\text{s-nec}_d(\mathcal{L})$ and the algorithm of [7] give efficient algorithms to solve any (σ, ρ) -DOMINATING SET problem, with parameters tree-width, clique-width, (\mathbb{Q}) -rank-width, and mim-width. The running times of these algorithms are given in Table 1.

■ **Table 1** Upper bounds on $\text{s-nec}_d(\mathcal{L})^{O(1)} \cdot n^{O(1)}$ with \mathcal{L} a layout and d a constant.

Tree-width	Clique-width	Rank-width	\mathbb{Q} -rank-width	Mim-width
$2^{O(k)} \cdot n^{O(1)}$	$2^{O(k)} \cdot n^{O(1)}$	$2^{O(k^2)} \cdot n^{O(1)}$	$2^{O(k \log(k))} \cdot n^{O(1)}$	$n^{O(k)}$

Our contributions. In this paper, we design a framework based on the 1-neighbor equivalence (presented in Section 3) and using some ideas of the rank-based approach of [5] to design efficient algorithms for many problems involving a connectivity constraint. This framework provides tools to reduce the size of the sets of partial solutions we compute at each step of a dynamic programming algorithm. We prove that many ad-hoc algorithms for these problems can be unified into a single algorithm that is almost the same as the one from [7] computing a dominating set.

In Section 4, we use our framework to design an algorithm that, given a rooted layout \mathcal{L} , solves any connectivity variant (a solution must induce a connected graph) of a (σ, ρ) -DOMINATING SET problem. This includes some well-known problems such as CONNECTED DOMINATING SET, CONNECTED VERTEX COVER or NODE WEIGHTED STEINER TREE. The running time of our algorithm is polynomial in n and $\text{s-nec}_d(\mathcal{L})$, with d a constant that depends on σ and ρ . Consequently, each connectivity variant of a (σ, ρ) -DOMINATING SET problem admits algorithms with the running times given in Table 1.

In Section 5, we introduce some new concepts to deal with acyclicity. We use these concepts to deal with the *AC* variants² (a solution must induce a tree) of (σ, ρ) -DOMINATING SET problems. Both MAXIMUM INDUCED TREE and LONGEST INDUCED PATH are *AC* variants of (σ, ρ) -DOMINATING SET problems. We prove that there exist algorithms that solve these *AC* variants in the running times given in Table 1. To obtain these results, we rely heavily on the d -neighbor equivalence. However, we were not able to provide an algorithm whose running time is polynomial in n and $\text{s-nec}_d(\mathcal{L})$ for some constant d . Instead, we provide an algorithm whose behavior depends slightly on each width measure considered in Table 1. We moreover prove that we can modify slightly this algorithm to solve any acyclic variant (a solution must induce a forest) of a (σ, ρ) -DOMINATING SET problem. In particular, this shows that we can use the algorithm for MAXIMUM INDUCED TREE to solve the FEEDBACK VERTEX SET problem.

Up to a constant in the exponent, the running times of our algorithms and their algorithmic consequences match those of the best known algorithms for basic problems such as VERTEX COVER and DOMINATING SET [7, 19]. Surprisingly, our result reveal that the d -neighbor equivalence relation can be used for problems which are not based on local constraints. This highlights the importance and the generalizing power of this concept on many width measures.

We conclude in Section 6 and state our theorem for the computation of MAX CUT – a problem which is $W[1]$ -hard parameterized by clique-width – whose running time is polynomial in n and the n -neighbor width of a given rooted layout. This algorithm gives the best known algorithms parameterized by clique-width, \mathbb{Q} -rank-width and rank-width. It is worth mentioning that contrary to the algorithm for MAX CUT given in [11], there is no need to assume that the graph is given with a clique-width expression as our algorithm can be parameterized by \mathbb{Q} -rank-width, which is always smaller than clique-width and for which also a fast FPT $(3k + 1)$ -approximation algorithm exists [20].

² *AC* stands for “acyclic and connected”.

Relation to previous works. Our framework can be used on tree-decomposition to obtain $2^{O(k)} \cdot n^{O(1)}$ time algorithms parameterized by tree-width for the variants of (σ, ρ) -DOMINATING SET problems. Indeed, given a vertex separator S of size k , the number of d -neighbor equivalence classes over S (resp. $V(G) \setminus S$) is upper bounded by 2^k (resp. $(d+1)^k$). For this reason, we can consider our framework as a generalization of the rank-based approach of [5]. Our framework generalizes also the clique-width adaptation of the rank-based approach used in [3] to obtain $2^{O(k)} \cdot n$ time algorithms, k being the clique-width of a given decomposition, for CONNECTED (σ, ρ) -DOMINATING SET problem and FEEDBACK VERTEX SET. However, the constant in the running time of the algorithms in [3, 5] are better than those of our algorithms. Indeed, our approach is based on a more general parameter and is not optimized neither for tree-width nor clique-width. Our framework simplifies the algorithms in [3, 5] because contrary to [3, 5] we do not use weighted partitions to encode the partial solutions. Consequently, the definitions of the dynamic programming tables and the computational steps of our algorithms are simpler than those in [3, 5]. This is particularly true for FEEDBACK VERTEX SET where the use of weighted partitions to encode the partial solutions in [3] implies to take care of many technical details concerning the acyclicity.

The results we obtain simplify the $2^{O(k^2)} \cdot n^{O(1)}$ time algorithm parameterized by rank-width for FEEDBACK VERTEX SET from [12], and the $n^{O(k)}$ time algorithms parameterized by mim-width for FEEDBACK VERTEX SET and LONGEST INDUCED PATH from [14, 15].

Concerning mim-width, we provide unified polynomial-time algorithms for the considered problems for all well-known graph classes having bounded mim-width and for which a layout of bounded mim-width can be computed in polynomial time [1] (*e.g.*, interval graphs, circular arc graphs, permutation graphs, Dilworth- k graphs and k -polygon graphs for all fixed k). Notice that we also generalize one of the results from [17] proving that the CONNECTED VERTEX COVER problem is solvable in polynomial time for circular arc graphs.

It is worth noticing that the approach used in [9] called Cut & Count can also be generalized to the d -neighbor-width for any CONNECTED (σ, ρ) -DOMINATING SET problem with more or less the same arguments used in this paper (see the PhD thesis [2]).

2 Preliminaries

The size of a set V is denoted by $|V|$ and its power set is denoted by 2^V . We write $A \setminus B$ for the set difference of A from B . We denote by \mathbb{N} the set of non-negative integers and by \mathbb{N}^+ the set $\mathbb{N} \setminus \{0\}$. We let $\min(\emptyset) := +\infty$ and $\max(\emptyset) := -\infty$. Let V be a finite set. A set function $f : 2^V \rightarrow \mathbb{N}$ is *symmetric* if, for all $S \subseteq V$, we have $f(S) = f(V \setminus S)$.

Graphs. Our graph terminology is standard, and we refer to [10]. The vertex set of a graph G is denoted by $V(G)$ and its edge set by $E(G)$. For every vertex set $X \subseteq V(G)$, when the underlying graph is clear from context, we denote by \overline{X} , the set $V(G) \setminus X$. An edge between two vertices x and y is denoted by xy or yx . The set of vertices that is adjacent to x is denoted by $N_G(x)$. For a set $U \subseteq V(G)$, we define $N_G(U) := \bigcup_{x \in U} N_G(x)$. If the underlying graph is clear, then we may remove G from the subscript.

The subgraph of G induced by a subset X of its vertex set is denoted by $G[X]$. For $X, Y \subseteq V(G)$, we denote by $G[X, Y]$ the bipartite graph with vertex set $X \cup Y$ and edge set $\{xy \in E(G) : x \in X \text{ and } y \in Y\}$. Moreover, we denote by $M_{X,Y}$ the adjacency matrix between X and Y , *i.e.*, the (X, Y) -matrix such that $M_{X,Y}[x, y] = 1$ if $y \in N(x)$ and 0

otherwise. For a graph G , we denote by $\text{cc}(G)$ the partition $\{V(C) : C \text{ is a connected component of } G\}$. For two subsets \mathcal{A} and \mathcal{B} of $2^{V(G)}$, we define the *merging* of \mathcal{A} and \mathcal{B} , denoted by $\mathcal{A} \otimes \mathcal{B}$, as

$$\mathcal{A} \otimes \mathcal{B} := \begin{cases} \emptyset & \text{if } \mathcal{A} = \emptyset \text{ or } \mathcal{B} = \emptyset, \\ \{X \cup Y : X \in \mathcal{A} \text{ and } Y \in \mathcal{B}\} & \text{otherwise.} \end{cases}$$

Let $X \subseteq V(G)$. A *consistent cut* of X is an ordered bipartition (X_1, X_2) of X such that $N(X_1) \cap X_2 = \emptyset$. We denote by $\text{ccut}(X)$ the set of all consistent cuts of X .

The d -neighbor equivalence relation. Let G be a graph. The following definition is from [7]. Let $A \subseteq V(G)$ and $d \in \mathbb{N}^+$. Two subsets X and Y of A are *d -neighbor equivalent* w.r.t. A , denoted by $X \equiv_A^d Y$, if $\min(d, |X \cap N(u)|) = \min(d, |Y \cap N(u)|)$ for all $u \in \overline{A}$. It is not hard to check that \equiv_A^d is an equivalence relation.

For all $d \in \mathbb{N}^+$, we let $\text{nec}_d : 2^{V(G)} \rightarrow \mathbb{N}$ where, for all $A \subseteq V(G)$, $\text{nec}_d(A)$ is the number of equivalence classes of \equiv_A^d . Notice that while nec_1 is a symmetric function [16, Theorem 1.2.3], nec_d is not necessarily symmetric for $d \geq 2$.

In order to manipulate the equivalence classes of \equiv_A^d , one needs to compute a representative for each equivalence class in polynomial time. This is achieved with the following notion of a representative. Let G be a graph with an arbitrary ordering of $V(G)$ and let $A \subseteq V(G)$. For each $X \subseteq A$, let us denote by $\text{rep}_A^d(X)$ the lexicographically smallest set $R \subseteq A$ such that $|R|$ is minimized and $R \equiv_A^d X$. Moreover, we denote by \mathcal{R}_A^d the set $\{\text{rep}_A^d(X) : X \subseteq A\}$. It is worth noticing that the empty set always belongs to \mathcal{R}_A^d , for all $A \subseteq V(G)$ and $d \in \mathbb{N}^+$. Moreover, we have $\mathcal{R}_{V(G)}^d = \mathcal{R}_\emptyset^d = \{\emptyset\}$ for all $d \in \mathbb{N}^+$. In order to compute \mathcal{R}_A^d , we use the following lemma.

► **Lemma 1** ([7]). *For every $A \subseteq V(G)$ and $d \in \mathbb{N}^+$, one can compute in time $O(\text{nec}_d(A) \cdot \log(\text{nec}_d(A)) \cdot |V(G)|^2)$, the sets \mathcal{R}_A^d and a data structure, that given a set $X \subseteq A$, computes $\text{rep}_A^d(X)$ in time $O(\log(\text{nec}_d(A)) \cdot |A| \cdot |V(G)|)$.*

Graph width measures. A *rooted binary tree* is a binary tree with a distinguished vertex called the *root*. Since we manipulate at the same time graphs and trees representing them, the vertices of trees will be called *nodes*. A *rooted layout* of G is a pair $\mathcal{L} = (T, \delta)$ of a rooted binary tree T and a bijective function δ between $V(G)$ and the leaves of T . For each node x of T , let L_x be the set of all the leaves l of T such that the path from the root of T to l contains x . We denote by $V_x^\mathcal{L}$ the set of vertices that are in bijection with L_x , i.e., $V_x^\mathcal{L} := \{v \in V(G) : \delta(v) \in L_x\}$. When \mathcal{L} is clear from the context, we may remove \mathcal{L} from the superscript.

All the width measures dealt with in this paper are special cases of the following one, the difference being in each case the used set function. Given a set function $f : 2^{V(G)} \rightarrow \mathbb{N}$ and a rooted layout $\mathcal{L} = (T, \delta)$, the f -width of a node x of T is $f(V_x^\mathcal{L})$ and the f -width of (T, δ) , denoted by $f(T, \delta)$ (or $f(\mathcal{L})$), is $\max\{f(V_x^\mathcal{L}) : x \in V(T)\}$. Finally, the f -width of G is the minimum f -width over all rooted layouts of G .

For a graph G , we let $\text{s-nec}_d, \text{mw}, \text{mim}, \text{rw}, \text{rw}_\mathbb{Q}$ be functions from $2^{V(G)}$ to \mathbb{N} such that for every $A \subseteq V(G)$, $\text{s-nec}_d(A) = \max\{\text{nec}_d(A), \text{nec}_d(\overline{A})\}$, $\text{mw}(A)$ is the cardinality of $\{N(v) \cap \overline{A} : v \in A\}$, $\text{mim}(A)$ is the size of a maximum induced matching of the graph $G[A, \overline{A}]$ and $\text{rw}(A)$ (resp. $\text{rw}_\mathbb{Q}(A)$) is the rank over $GF(2)$ (resp. \mathbb{Q}) of the matrix $M_{A, \overline{A}}$. The *d -neighbor-width*, *module-width*, *mim-width*, *rank-width* and *\mathbb{Q} -rank-width* of G , are respectively, its s-nec_d -width, mw -width, mim -width, rw -width and $\text{rw}_\mathbb{Q}$ -width [23].

Observe that, for every graph G , $\text{mw}(G) \leq \text{cw}(G) \leq 2\text{mw}(G)$ where $\text{cw}(G)$ is the clique-width of G [21], and one can moreover translate, in time at most $O(n^2)$, a given decomposition into the other one with width at most the given bounds.

In the following, we fix G an n -vertex graph, (T, δ) a rooted layout of G , and $w : V(G) \rightarrow \mathbb{Q}$ a weight function over the vertices of G . We also assume that $V(G)$ is ordered.

3 Representative sets

In this section, we define a notion of representativity between sets of partial solutions *w.r.t.* the connectivity. Our notion of representativity is defined *w.r.t.* some node x of T and the 1-neighbor equivalence class of some set $R' \subseteq \overline{V}_x$. In our algorithms, R' will always belong to $\mathcal{R}_{\overline{V}_x}^d$ for some $d \in \mathbb{N}^+$. Our algorithms compute a set of partial solutions for each $R' \in \mathcal{R}_{\overline{V}_x}^d$. The partial solutions computed for R' will be completed with sets equivalent to R' *w.r.t.* $\equiv_{\overline{V}_x}^d$. Intuitively, the R' 's represent some expectation about how we will complete our sets of partial solutions. For the connectivity and the domination, $d = 1$ is enough but if we need more information for some reasons (for example the (σ, ρ) -domination or the acyclicity), we may take $d > 1$. This is not a problem as the d -neighbor equivalence class of R' is included in the 1-neighbor equivalence class of R' . Hence, in this section, we fix a node x of T and a set $R' \subseteq \overline{V}_x$ to avoid to overload the statements by the sentence “let x be a node of T and $R' \subseteq \overline{V}_x$ ”. We let $\text{opt} \in \{\min, \max\}$; if we want to solve a maximization (or minimization) problem, we use $\text{opt} = \max$ (or $\text{opt} = \min$). We use it also, as here, in the next sections.

► **Definition 2** ((x, R') -representativity). *For every $\mathcal{A} \subseteq 2^{V(G)}$ and $Y \subseteq V(G)$, we define $\text{best}(\mathcal{A}, Y) := \text{opt}\{w(X) : X \in \mathcal{A} \text{ and } G[X \cup Y] \text{ is connected}\}$.*

Let $\mathcal{A}, \mathcal{B} \subseteq 2^{V_x}$. We say that \mathcal{B} (x, R') -represents \mathcal{A} if, for every $Y \subseteq \overline{V}_x$ such that $Y \equiv_{\overline{V}_x}^1 R'$, we have $\text{best}(\mathcal{A}, Y) = \text{best}(\mathcal{B}, Y)$.

Notice that the (x, R') -representativity is an equivalence relation. The set \mathcal{A} is meant to represent a set of partial solutions of $G[V_x]$ associated with R' which have been computed. If a $\mathcal{B} \subseteq \mathcal{A}$ (x, R') -represents \mathcal{A} , then we can safely substitute \mathcal{A} by \mathcal{B} because the quality of the output of the dynamic programming algorithm will remain the same. Indeed, for every subset Y of \overline{V}_x such that $Y \equiv_{\overline{V}_x}^1 R'$, an optimum solution obtained by the union of a partial solution in \mathcal{A} and Y will have the same weight as an optimum solution obtained from the union of a set in \mathcal{B} and Y .

The following theorem presents the main tool of our framework: a function `reduce` that, given a set of partial solutions \mathcal{A} , outputs a subset of \mathcal{A} that (x, R') -represents \mathcal{A} and whose size is upper bounded by $\text{s-nec}_1(\mathcal{L})^2$. To design this function, we use ideas from the rank-based approach of [5]. That is, we define a small matrix \mathcal{C} with $|\mathcal{A}|$ rows and $\text{s-nec}_1(V_x)^2$ columns. Then, we show that a basis of maximum weight of the row space of \mathcal{C} corresponds to an (x, R') -representative set of \mathcal{A} . Since \mathcal{C} has $\text{s-nec}_1(\mathcal{L})^2$ columns, the size of a basis of \mathcal{C} is smaller than $\text{s-nec}_1(\mathcal{L})^2$. By calling `reduce` after each computational step, we keep the sizes of the sets of partial solutions polynomial in $\text{s-nec}_1(\mathcal{L})$.

In order to compute a small (x, R') -representative set of a set $\mathcal{A} \subseteq 2^{V_x}$, the following theorem requires that the sets in \mathcal{A} are pairwise equivalent *w.r.t.* $\equiv_{\overline{V}_x}^1$. This is useful since in our algorithm we classify our sets of partial solutions with respect to this property. We need this to guarantee that the partial solutions computed for R' will be completed with sets equivalent to R' *w.r.t.* $\equiv_{\overline{V}_x}^d$. However, if one wants to compute a small (x, R') -representative set of a set \mathcal{A} that does not respect this property, then it is enough to compute an (x, R') -representative set for each 1-neighbor equivalence class of \mathcal{A} . The union of these (x, R') -representative sets is an (x, R') -representative set of \mathcal{A} . In the following, ω is the matrix multiplication exponent.

► **Theorem 3.** *Let $R \in \mathcal{R}_{V_x}^1$ and $R' \subseteq \overline{V_x}$. Then, there exists an algorithm `reduce` that, given $\mathcal{A} \subseteq 2^{V_x}$ such that $X \equiv_{V_x}^1 R$ for all $X \in \mathcal{A}$, outputs in time $O(|\mathcal{A}| \cdot \text{nec}_1(V_x)^{2(\omega-1)} \cdot n^2)$ a subset $\mathcal{B} \subseteq \mathcal{A}$ such that \mathcal{B} (x, R') -represents \mathcal{A} and $|\mathcal{B}| \leq \text{nec}_1(V_x)^2$.*

Sketch of proof. We assume w.l.o.g. that $\text{opt} = \max$ (the case $\text{opt} = \min$ is symmetric). If $R' \equiv_{V_x}^1 \emptyset$, then, from Definition 2, it is enough to output $\{X\}$ where X is a set in \mathcal{A} of maximum weight such that $G[X]$ is connected.

Assume that R' is not equivalent to \emptyset w.r.t. $\equiv_{V_x}^1$. Observe that, for every $X \in \mathcal{A}$ which admits a connected component C with $N(C) \cap R' = \emptyset$, the graph $G[X \cup Y]$ is not connected for every $Y \equiv_{V_x}^1 R'$. Thus, we can safely remove from \mathcal{A} all such sets, this can be done in time $|\mathcal{A}| \cdot n^2$. Let \mathcal{D} be the set of all subsets Y of $\overline{V_x}$ such that $Y \equiv_{V_x}^1 R'$ and, for all $C \in \text{cc}(Y)$, we have $N(C) \cap R' \neq \emptyset$. It is easy to check that the sets in $2^{\overline{V_x}} \setminus \mathcal{D}$ do not matter in the (x, R') -representativity: they will not give a solution with a set in \mathcal{A} .

For every $Y \in \mathcal{D}$, we let v_Y be one fixed vertex of Y . In the following, we denote by \mathfrak{R} the set $\{(R'_1, R'_2) \in \mathcal{R}_{V_x}^1 \times \mathcal{R}_{V_x}^1\}$. Let \mathcal{C} , and $\overline{\mathcal{C}}$ be, respectively, an $(\mathcal{A}, \mathfrak{R})$ -matrix and an $(\mathfrak{R}, \mathcal{D})$ -matrix such that

$$\mathcal{C}[X, (R'_1, R'_2)] := \begin{cases} 1 & \text{if } \exists (X_1, X_2) \in \text{ccut}(X) \text{ such that } N(X_1) \cap R'_2 = \emptyset \wedge N(X_2) \cap R'_1 = \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

$$\overline{\mathcal{C}}[(R'_1, R'_2), Y] := \begin{cases} 1 & \text{if } \exists (Y_1, Y_2) \in \text{ccut}(Y) \text{ such that } v_Y \in Y_1, Y_1 \equiv_{V_x}^1 R'_1, \text{ and } Y_2 \equiv_{V_x}^1 R'_2, \\ 0 & \text{otherwise.} \end{cases}$$

Owing to the partial solutions we have removed from \mathcal{A} and the definition of \mathcal{D} , we can prove that a basis of maximum weight of the row space of \mathcal{C} is an (x, R') -representative set of \mathcal{A} . This follows from the fact that $(\mathcal{C} \cdot \overline{\mathcal{C}})[X, Y]$ equals the number of consistent cuts (W_1, W_2) in $\text{ccut}(X \cup Y)$ such that $v_Y \in W_1$. Consequently, $(\mathcal{C} \cdot \overline{\mathcal{C}})[X, Y] = 2^{|\text{cc}(G[X \cup Y])| - 1}$ and thus $(\mathcal{C} \cdot \overline{\mathcal{C}})[X, Y]$ is odd if and only if $G[X \cup Y]$ is connected. The running time of `reduce` and the size of `reduce`(\mathcal{A}) follows from the size of \mathcal{C} (i.e. $|\mathcal{A}| \cdot \text{nec}_1(V_x)^2$) and the fact that both \mathcal{C} and a basis of maximum weight of \mathcal{C} are easy to compute. ◀

Now to boost up a dynamic programming algorithm P on some rooted layout (T, δ) of G , we can use the function `reduce` to keep the size of the sets of partial solutions bounded by $\text{s-nec}_1(T, \delta)^2$. We call P' the algorithm obtained from P by calling the function `reduce` at every step of computation. We can assume that the set of partial solutions \mathcal{A}_r computed by P and associated with the root r of (T, δ) contains an optimal solution (this will be the cases in our algorithms). To prove the correctness of P' , we need to prove that \mathcal{A}'_r (r, \emptyset) -represents \mathcal{A}_r where \mathcal{A}'_r is the set of partial solutions computed by P' and associated with r . For doing so, we need to prove that at each step of the algorithm the operations we use preserve the (x, R') -representativity. The following fact states that we can use the union without restriction, it follows directly from Definition 2 of (x, R') -representativity.

► **Fact 4.** *If \mathcal{B} and \mathcal{D} (x, R') -represents, respectively, \mathcal{A} and \mathcal{C} , then $\mathcal{B} \cup \mathcal{D}$ (x, R') -represents $\mathcal{A} \cup \mathcal{C}$.*

The second operation we use in our dynamic programming algorithms is the merging operator \otimes . In order to safely use it, we need the following notion of compatibility that just tells which partial solutions from V_a and V_b can be joined to possibly form a partial solution in V_x . (It was already used in [7] without naming it.)

► **Definition 5** (d - (R, R') -compatibility). Suppose that x is an internal node of T with a and b as children. Let $d \in \mathbb{N}^+$ and $R \in \mathcal{R}_{V_x}^d$. We say that $(A, A') \in \mathcal{R}_{V_a}^d \times \mathcal{R}_{V_a}^d$ and $(B, B') \in \mathcal{R}_{V_b}^d \times \mathcal{R}_{V_b}^d$ are d - (R, R') -compatible if we have (1) $A \cup B \equiv_{V_x}^d R$, (2) $A' \equiv_{V_a}^d B \cup R'$, and (3) $B' \equiv_{V_b}^d A \cup R'$.

► **Lemma 6.** Suppose that x is an internal node of T with a and b as children. Let $d \in \mathbb{N}^+$ and $R \in \mathcal{R}_{V_x}^d$. Let $(A, A') \in \mathcal{R}_{V_a}^d \times \mathcal{R}_{V_a}^d$ and $(B, B') \in \mathcal{R}_{V_b}^d \times \mathcal{R}_{V_b}^d$ that are d - (R, R') -compatible. Let $\mathcal{A} \subseteq 2^{V_a}$ such that, for all $X \in \mathcal{A}$, we have $X \equiv_{V_a}^d A$, and let $\mathcal{B} \subseteq 2^{V_b}$ such that, for all $W \in \mathcal{B}$, we have $W \equiv_{V_b}^d B$. If $\mathcal{A}' \subseteq \mathcal{A}$ (a, A')-represents \mathcal{A} and $\mathcal{B}' \subseteq \mathcal{B}$ (b, B')-represents \mathcal{B} , then $\mathcal{A}' \otimes \mathcal{B}'$ (x, R')-represents $\mathcal{A} \otimes \mathcal{B}$.

4 Connected (Co)- (σ, ρ) -Dominating Sets

Let σ and ρ be two (non-empty) finite or co-finite subsets of \mathbb{N} . We say that a subset D of $V(G)$ (σ, ρ)-dominates a subset $U \subseteq V(G)$ if, for every vertex $u \in U \cap D$, we have $|N(u) \cap D| \in \sigma$, and, for every vertex $u \in U \setminus D$, we have $|N(u) \cap D| \in \rho$. A subset D of $V(G)$ is a (σ, ρ)-dominating set (resp. co- (σ, ρ) -dominating set) if D (resp. $V(G) \setminus D$) (σ, ρ)-dominates $V(G)$. A problem is a CONNECTED (σ, ρ) -DOMINATING SET if it consists in finding a connected (σ, ρ) -dominating set of maximum (or minimum) weight. Similarly, one can define the CONNECTED CO- (σ, ρ) -DOMINATING SET problems. Examples are CONNECTED PERFECT DOMINATING SET and CONNECTED VERTEX-COVER³.

Let $d(\mathbb{N}) := 0$, and for a finite or co-finite subset μ of \mathbb{N} , let $d(\mu) := \min(\max(\mu), \max(\mathbb{N} \setminus \mu)) + 1$. Let $d := \max\{1, d(\sigma), d(\rho)\}$. The definition of d is motivated by the following lemma.

► **Lemma 7** ([7]). Let $A \subseteq V(G)$. Let $X \subseteq A$ and $Y, Y' \subseteq \bar{A}$ such that $Y \equiv_A^d Y'$. Then $(X \cup Y)$ (σ, ρ)-dominates A if and only if $(X \cup Y')$ (σ, ρ)-dominates A .

In this section, we present an algorithm that computes a maximum (or minimum) connected (σ, ρ) -dominating set with a graph G and a layout (T, δ) as inputs. Its running time is $O(\text{s-nec}_d(T, \delta)^{O(1)} \cdot n^3)$. The same algorithm, with some little modifications, will be able to find a minimum Steiner tree or a maximum (or minimum) connected co- (σ, ρ) -dominating set as well.

To deal with the local constraint, *i.e.*, the (σ, ρ) -domination, we use the ideas of Bui-Xuan et al. [7]. For every $x \in V(T)$ and all pairs $(R, R') \in \mathcal{R}_{V_x}^d \times \mathcal{R}_{V_x}^d$, we let $\mathcal{A}_x[R, R'] := \{X \subseteq V_x : X \equiv_{V_x}^d R \text{ and } X \cup R' \text{ } (\sigma, \rho)\text{-dominates } V_x\}$. To compute a maximum (or minimum) (σ, ρ) -dominating set, Bui-Xuan et al. [7] proved that it is enough to compute, for each node $x \in V(T)$ and each pair (R, R') , a partial solution X in $\mathcal{A}_x[R, R']$ of maximum (or minimum) weight. Indeed, by Lemma 7, if a partial solution $X \in \mathcal{A}_x[R, R']$ could be completed into a (σ, ρ) -dominating set of G with a set $Y \equiv_{V_x}^d R'$, then it is the case for every partial solution in $\mathcal{A}_x[R, R']$. To deal with the connectivity constraint, for each node x of $V(T)$ and each pair (R, R') , our algorithm will compute a set $\mathcal{D}_x[R, R']$ that satisfies the following invariant.

Invariant. For every $(R, R') \in \mathcal{R}_{V_x}^d \times \mathcal{R}_{V_x}^d$, the set $\mathcal{D}_x[R, R']$ is a subset of $\mathcal{A}_x[R, R']$ of size at most $\text{s-nec}_1(T, \delta)^2$ that (x, R') -represents $\mathcal{A}_x[R, R']$.

Notice that, by the definition of $\mathcal{A}_r[\emptyset, \emptyset]$ (r being the root of T) and the definition of (x, R') -representativity, if G admits a connected (σ, ρ) -dominating set, then $\mathcal{D}_r[\emptyset, \emptyset]$ must contain a maximum (or minimum) connected (σ, ρ) -dominating set.

³ More can be found in [7, Table 1] by adding a connectivity constraint to the sets or their complements.

We compute the tables \mathcal{D}_x 's by a usual bottom-up dynamic programming algorithm, starting at the leaves of T . The computational steps are trivial for the leaves, and for each internal node x with children a and b , and for each $(R, R') \in \mathcal{R}_{V_x}^d \times \mathcal{R}_{V_x}^d$ we let $\mathcal{D}_x[R, R']$ be an (x, R') -representative set of $\left(\bigcup_{(A, A'), (B, B') \text{ } d\text{-}(R, R')\text{-compatible}} \mathcal{A}_a[A, A'] \otimes \mathcal{A}_b[B, B']\right)$ computed with the function `reduce` defined in Section 3. This guarantees that each set $\mathcal{D}_x[R, R']$ contains at most $\mathfrak{s}\text{-nec}_1(T, \delta)^2$ partial solutions. Thanks to Lemma 6, we can state.

► **Theorem 8.** *There exists an algorithm that, given an n -vertex graph G and a rooted layout (T, δ) of G , computes a maximum (or minimum) connected (σ, ρ) -dominating set in time $O(\mathfrak{s}\text{-nec}_d(T, \delta)^3 \cdot \mathfrak{s}\text{-nec}_1(T, \delta)^{2(\omega+1)} \cdot \log(\mathfrak{s}\text{-nec}_d(T, \delta)) \cdot n^3)$ with $d := \max\{1, d(\sigma), d(\rho)\}$.*

As a corollary, we can solve in time $\mathfrak{s}\text{-nec}_1(T, \delta)^{O(1)} \cdot n^3$ the NODE-WEIGHTED STEINER TREE problem that asks, given a subset of vertices $K \subseteq V(G)$ called *terminals*, a subset T of minimum weight such that $K \subseteq T \subseteq V(G)$ and $G[T]$ is connected.

► **Corollary 9.** *There exists an algorithm that, given an n -vertex graph G , a subset $K \subseteq V(G)$, and a rooted layout (T, δ) of G , computes a minimum node-weighted Steiner tree for (G, K) in time $O(\mathfrak{s}\text{-nec}_1(T, \delta)^{2\omega+5} \cdot \log(\mathfrak{s}\text{-nec}_1(T, \delta)) \cdot n^3)$.*

Observe that Corollary 9 simplify and generalize the algorithm from [5] for (EDGE-WEIGHTED) STEINER TREE. Indeed the incidence graph of a graph of tree-width k has tree-width at most $k + 1$, and one can reduce the computation of a edge-weighted Steiner tree on a graph to the computation of a node-weighted Steiner tree on its incidence graph.

With few modifications of the algorithm from Theorem 8, we can state the following.

► **Corollary 10.** *There exists an algorithm that, given an n -vertex graph G and a rooted layout (T, δ) of G , computes a maximum (or minimum) connected $\text{co-}(\sigma, \rho)$ -dominating set in time $O(\mathfrak{s}\text{-nec}_d(T, \delta)^3 \cdot \mathfrak{s}\text{-nec}_1(T, \delta)^{2\omega+5} \cdot \log(\mathfrak{s}\text{-nec}_d(T, \delta)) \cdot n^3)$ with $d := \max\{1, d(\sigma), d(\rho)\}$.*

5 Acyclic variants of (Connected) (σ, ρ) -Dominating Set

We call AC- (σ, ρ) -DOMINATING SET (resp. ACYCLIC (σ, ρ) -DOMINATING SET) the family of problems which consists in finding a subset $X \subseteq V(G)$ of maximum (or minimum) weight such that X is a (σ, ρ) -dominating set of G and $G[X]$ is a tree (resp. a forest). Examples are LONGEST INDUCED PATH and FEEDBACK VERTEX SET.

In this section, we present an algorithm that solves any AC- (σ, ρ) -DOMINATING SET problem. Unfortunately, we were not able to obtain an algorithm whose running time is polynomial in n and the d -neighbor-width of the given layout (for some constant d). But, for the other parameters, by using their respective properties, we get the running time presented in Table 1. Moreover, we show, via a polynomial reduction, that we can use our algorithm for AC- (σ, ρ) -DOMINATING SET problems (with some modifications) to solve any ACYCLIC (σ, ρ) -DOMINATING SET problem.

Let us first explain why we cannot use the same trick as in [5] on the algorithms of Section 4 to ensure the acyclicity, that is classifying the partial solutions X – associated with a node $x \in V(T)$ – with respect to $|X|$ and $|E(G[X])|$. Indeed, for two sets $X, W \subseteq V_x$ with $|X| = |W|$ and $|E(G[X])| = |E(G[W])|$, we have $|E(G[X \cup Y])| = |E(G[W \cup Y])|$, for all $Y \subseteq \overline{V_x}$, if and only if $X \equiv_{V_x}^n W$. Hence, the trick used in [5] would imply to classify the partial solutions with respect to their n -neighbor equivalence class. But, the upper bounds we have on $\text{nec}_n(V_x)$ with respect to module-width, (\mathbb{Q}) -rank-width would lead to an XP algorithm.

17:10 Connectivity and Acyclicity Constraints Versus d -Neighbor Equivalence

In the following, we introduce some new concepts that extends the framework designed in Section 3 in order to manage acyclicity. All along, we give intuitions on these concepts through a concrete example: MAXIMUM INDUCED TREE. Finally, we present the algorithms for the AC- (σ, ρ) -DOMINATING SET problems and the algorithms for ACYCLIC (σ, ρ) -DOMINATING SET problems. We start by defining a new notion of representativity to deal with the acyclicity constraint. This new notion of representativity is defined *w.r.t.* to the 2-neighbor equivalence class of a set $R' \subseteq \overline{V_x}$. We consider 2-neighbor equivalence classes instead of 1-neighbor equivalence classes in order to manage the acyclicity (see the following explanations). Similarly to Section 3, every concept introduced in this section is defined with respect to a node x of T and a set $R' \subseteq \overline{V_x}$. To simplify this section, we fix a node x of T and $R' \subseteq \overline{V_x}$. In our algorithm, R' will always belong to $\mathcal{R}_{\overline{V_x}}^d$ for some $d \in \mathbb{N}^+$ with $d \geq 2$. For MAXIMUM INDUCED TREE $d = 2$ is enough and in general, we use $d := \max\{2, d(\sigma), d(\rho)\}$. The following definition extends Definition 2 of Section 3 to deal with the acyclicity.

► **Definition 11** ((x, R') -acy-representativity). *For every $\mathcal{A} \subseteq 2^{V(G)}$ and $Y \subseteq V(G)$, we define $\text{best}(\mathcal{A}, Y)^{\text{acy}} := \text{opt}\{\text{w}(X) : X \in \mathcal{A} \text{ and } G[X \cup Y] \text{ is a tree}\}$.*

Let $\mathcal{A}, \mathcal{B} \subseteq 2^{V_x}$. We say that \mathcal{B} (x, R') -acy-represents \mathcal{A} if, for every $Y \subseteq \overline{V_x}$ such that $Y \equiv_{\overline{V_x}}^2 R'$, we have $\text{best}^{\text{acy}}(\mathcal{A}, Y) = \text{best}^{\text{acy}}(\mathcal{B}, Y)$.

In order to compute a maximum induced tree, we design an algorithm similar to those of Section 4. That is, for each $(R, R') \in \mathcal{R}_{V_x}^2 \times \mathcal{R}_{\overline{V_x}}^2$, our algorithm will compute a set $\mathcal{D}_x[R, R'] \subseteq 2^{V_x}$ that is an (x, R') -acy-representative set of small size of the set $\mathcal{A}_x[R] := \{X \subseteq V_x \text{ such that } X \equiv_{\overline{V_x}}^2 R\}$. This is sufficient to compute a maximum induced tree of G since we have $\mathcal{A}_r[\emptyset] = 2^{V(G)}$ with r the root of T . Thus, by Definition 11, any $(r, \emptyset)^{\text{acy}}$ -representative set of $\mathcal{A}_r[\emptyset]$ contains a maximum induced tree.

The key to compute the tables of our algorithm is a function that, given $\mathcal{A} \subseteq 2^{V_x}$, computes a small subset of \mathcal{A} that (x, R') -acy-represents \mathcal{A} . This function starts by removing from \mathcal{A} some sets that will never give a tree with a set $Y \equiv_{\overline{V_x}}^2 R'$. For doing so, we characterize the sets $X \in \mathcal{A}$ such that $G[X \cup Y]$ is a tree for some $Y \equiv_{\overline{V_x}}^2 R'$. The following gives a formal definition of these important and unimportant partial solutions.

► **Definition 12** (R' -important). *We say that $X \subseteq V_x$ is R' -important if there exists $Y \subseteq \overline{V_x}$ such that $Y \equiv_{\overline{V_x}}^2 R'$ and $G[X \cup Y]$ is a tree, otherwise, we say that X is R' -unimportant.*

By definition, any set obtained from a set \mathcal{A} by removing R' -unimportant sets is an (x, R') -acy-representative set of \mathcal{A} . The following lemma gives some necessary conditions on R' -important sets. It follows that any set which does not respect one of these conditions can safely be removed from \mathcal{A} . These conditions are the key to obtain the running times of Table 1. At this point, we need to introduce the following notations. For every $X \subseteq V_x$, we define $X^0 := \{v \in X : N(v) \cap R' = \emptyset\}$, $X^1 := \{v \in X : |N(v) \cap R'| = 1\}$, and $X^{2+} := \{v \in X : |N(v) \cap R'| \geq 2\}$. Notice that, for every $Y \equiv_{\overline{V_x}}^2 R'$ and $X \subseteq V_x$, the vertices in X^0 have no neighbor in Y , those in X^1 have exactly one neighbor in Y and those in X^{2+} have at least 2 neighbors in Y .

► **Lemma 13.** *If $X \subseteq V_x$ is R' -important, then $G[X]$ is a forest and the following properties are satisfied:*

1. *for every pair of distinct vertices a and b in X^{2+} , we have $N(a) \cap \overline{V_x} \neq N(b) \cap \overline{V_x}$,*
2. *$|X^{2+}|$ is upper bounded by $2\text{mim}(V_x)$, $2\text{rw}(V_x)$ and $2\text{rw}_{\mathbb{Q}}(V_x)$.*

In order to prove these two necessary conditions, we need the properties of the 2-neighbor equivalence relation. More precisely, we use the fact that, for all $X \subseteq V_x$ and $Y \equiv_{\overline{V_x}}^2 R'$, the vertices in X having at least two neighbors in Y corresponds exactly to those having

at least two neighbors in R' . These vertices play a major role in the acyclicity and the computation of representatives in the following sense. By removing from \mathcal{A} the sets that do not respect the two above properties, we are able to decompose \mathcal{A} into a small number of sets $\mathcal{A}_1, \dots, \mathcal{A}_t$ such that an (x, R') -representative set of \mathcal{A}_i is an $(x, R')^{\text{acy}}$ -representative set of \mathcal{A}_i for each $i \in \{1, \dots, t\}$. We find an $(x, R')^{\text{acy}}$ -representative set of \mathcal{A} , by computing an (x, R') -representative set \mathcal{B}_i for each \mathcal{A}_i with the function `reduce`. This is sufficient because $\mathcal{B}_1 \cup \dots \cup \mathcal{B}_t$ is an $(x, R')^{\text{acy}}$ -representative set of \mathcal{A} . The following definition gives a characterization of the subsets of 2^{V_x} for which an (x, R') -representative set is also an $(x, R')^{\text{acy}}$ -representative set.

► **Definition 14.** We say that $\mathcal{A} \subseteq 2^{V_x}$ is R' -consistent if, for each $Y \subseteq \overline{V_x}$ such that $Y \equiv_{\overline{V_x}}^2 R'$, if there exists $W \in \mathcal{A}$ such that $G[W \cup Y]$ is a tree, then, for each $X \in \mathcal{A}$, either $G[X \cup Y]$ is a tree or $G[X \cup Y]$ is not connected.

The following lemma proves that an (x, R') -representative set of an R' -consistent set is also an $(x, R')^{\text{acy}}$ -representative set of this latter.

► **Lemma 15.** Let $\mathcal{A} \subseteq 2^{V_x}$. For all $\mathcal{D} \subseteq \mathcal{A}$, if \mathcal{A} is R' -consistent and \mathcal{D} (x, R') -represents \mathcal{A} , then \mathcal{D} $(x, R')^{\text{acy}}$ -represents \mathcal{A} .

The next lemma proves that, for each $f \in \{\text{mw}, \text{rw}, \text{rw}_{\mathbb{Q}}, \text{mim}\}$, we can decompose a set $\mathcal{A} \subseteq 2^{V_x}$ into a small collection $\{\mathcal{A}_1, \dots, \mathcal{A}_t\}$ of pairwise disjoint subsets of \mathcal{A} such that each \mathcal{A}_i is R' -consistent. Even though some parts of the proof are specific to each parameter, the ideas are roughly the same. First, we remove the sets X in \mathcal{A} that do not induce a forest. If $f = \text{mw}$, we remove the sets in \mathcal{A} that do not respect Condition (1) of Lemma 13, otherwise, we remove the sets that do not respect the upper bound associated with f from Condition (2) of Lemma 13. These sets can be safely removed as, by Lemma 13, they are R' -unimportant. After removing these sets, we obtain the decomposition of \mathcal{A} by taking the equivalence classes of some equivalence relation that is roughly the n -neighbor equivalence relation. Owing to the set of R' -unimportant sets we have removed from \mathcal{A} , we prove that the number of equivalence classes of this latter equivalence relation respects the upper bound associated with f that is described in Table 2.

► **Lemma 16.** Let $\mathcal{A} \subseteq 2^{V_x}$. For each $f \in \{\text{mw}, \text{rw}, \text{rw}_{\mathbb{Q}}, \text{mim}\}$, there exists a collection $\{\mathcal{A}_1, \dots, \mathcal{A}_t\}$ of pairwise disjoint subsets of \mathcal{A} computable in time $O(|\mathcal{A}| \cdot \mathcal{N}_f(T, \delta) \cdot n^2)$ such that (1) $\mathcal{A}_1 \cup \dots \cup \mathcal{A}_t$ $(x, R')^{\text{acy}}$ -represents \mathcal{A} , (2) \mathcal{A}_i is R' -consistent for each $i \in \{1, \dots, t\}$, and (3) $t \leq \mathcal{N}_f(T, \delta)$; where $\mathcal{N}_f(T, \delta)$ is the term defined in Table 2.

■ **Table 2** Upper bounds $\mathcal{N}_f(T, \delta)$ on the size of the decomposition of Lemma 16 for each $f \in \{\text{mw}, \text{rw}, \text{rw}_{\mathbb{Q}}, \text{mim}\}$.

f	mw	rw _Q	rw	mim
$\mathcal{N}_f(T, \delta)$	$2^{\text{mw}(T, \delta)} \cdot 2n$	$2^{\text{rw}_{\mathbb{Q}}(T, \delta) \log_2(2\text{rw}_{\mathbb{Q}}(T, \delta) + 1)} \cdot 2n$	$2^{2\text{rw}(T, \delta)^2} \cdot 2n$	$2n^{2\text{mim}(T, \delta) + 1}$

We are now ready to give an adaptation of Theorem 3 to the notion of $(x, R')^{\text{acy}}$ -representativity.

► **Theorem 17.** Let $R \in \mathcal{R}_{V_x}^2$. For each $f \in \{\text{mw}, \text{rw}, \text{rw}_{\mathbb{Q}}, \text{mim}\}$, there exists an algorithm `reducefacy` that, given a set \mathcal{A} such that $X \equiv_{V_x}^2 R$ for every $X \in \mathcal{A}$, outputs in time $O((\text{nec}_1(V_x)^{2(\omega-1)} + \mathcal{N}_f(T, \delta)) \cdot |\mathcal{A}| \cdot n^2)$, a subset $\mathcal{B} \subseteq \mathcal{A}$ such that \mathcal{B} $(x, R')^{\text{acy}}$ -represents \mathcal{A} and $|\mathcal{B}| \leq \mathcal{N}_f(T, \delta) \cdot \text{nec}_1(V_x)^2$.

17:12 Connectivity and Acyclicity Constraints Versus d -Neighbor Equivalence

Now, The algorithm for any AC- (σ, ρ) -DOMINATING SET problem is essentially the same as the algorithm from Theorem 8, except that we use $\text{reduce}_f^{\text{acy}}$ instead of reduce .

► **Theorem 18.** *For each $f \in \{\text{mw}, \text{rw}, \text{rw}_{\mathbb{Q}}, \text{mim}\}$, there exists an algorithm that, given an n -vertex graph G and a rooted layout (T, δ) of G , solves any AC- (σ, ρ) -DOMINATING SET problem, in time $O(\text{s-nec}_d(T, \delta)^3 \cdot \text{s-nec}_1(T, \delta)^{2(\omega+1)} \cdot \mathcal{N}_f(T, \delta)^2 \cdot \log(\text{s-nec}_d(T, \delta)) \cdot n^3)$, with $d := \max\{2, d(\sigma), d(\rho)\}$.*

By constructing for any graph G a graph G' such that the width measure of G' is linear in the width measure of G , and any optimum acyclic (σ, ρ) -dominating set of G corresponds to an optimum AC- (σ, ρ) -dominating set of G' and vice-versa, we obtain the following which allows for instance to compute a feedback vertex set in time $n^{O(c)}$, c the mim-width.

► **Theorem 19.** *For each $f \in \{\text{mw}, \text{rw}, \text{rw}_{\mathbb{Q}}, \text{mim}\}$, there exists an algorithm that, given an n -vertex graph G and a rooted layout (T, δ) of G , solves any ACYCLIC (σ, ρ) -DOMINATING SET problem in time $O(\text{s-nec}_d(T, \delta)^{O(1)} \cdot \mathcal{N}_f(T, \delta)^{O(1)} \cdot n^3)$ with $d := \max\{2, d(\sigma), d(\rho)\}$.*

6 Conclusion

Prior to this work, the d -neighbor-equivalence relation was used only for problems with a locally checkable property like DOMINATING SET [7, 13, 19]. We prove that the d -neighbor-equivalence relation can also be useful for problems with a connectivity constraint and an acyclicity constraint. Is this notion also useful for other kinds of problems? Can we use it for the problems which are unlikely to admit FPT algorithms parameterized by clique-width, \mathbb{Q} -rank-width or rank-width? This is the case for well-known problems such as HAMILTONIAN CYCLE, EDGE DOMINATING SET, and MAX CUT. The complexity of these problems parameterized by clique-width is well-known. Indeed, for each of these problems, we have an ad-hoc $n^{O(k)}$ time algorithm with k the clique-width of a given k -expression [4, 11]. On the other hand, little is known concerning rank-width and \mathbb{Q} -rank-width. For mim-width, we know that HAMILTONIAN CYCLE is para-NP-hard parameterized by the mim-width of a given rooted layout [15].

As these problems are W[1]-hard parameterized by clique-width, we cannot expect to rely only on the d -neighbor equivalence relation for d a constant. Maybe, we can avoid this dead-end by using the n -neighbor equivalence relation. In fact, we prove the following theorem for MAX CUT.

► **Theorem 20.** *There exists an algorithm that, given an n -vertex graph G and a rooted layout (T, δ) , solves MAX CUT in time $O(\text{s-nec}_n(T, \delta)^2 \cdot \log(\text{s-nec}_n(T, \delta)) \cdot n^3)$.*

Consequently, this theorem implies the existence of $n^{O(\text{mw}(\mathcal{L}))}$, $n^{O(\text{rw}_{\mathbb{Q}}(G))}$ and $n^{2^{O(\text{rw}(G))}}$ time algorithms for MAX CUT. Notice that, unless ETH fails, there are no $n^{o(\text{mw}(\mathcal{L}))}$ and $n^{o(\text{rw}_{\mathbb{Q}}(G))}$ time algorithms for MAX CUT [11].

References

- 1 Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theoret. Comput. Sci.*, 511:54–65, 2013. doi:10.1016/j.tcs.2013.01.011.
- 2 Benjamin Bergougnoux. *Matrix Decomposition and Algorithmic Application to (Hyper)Graphs*. PhD thesis, Université Clermont Auvergne, 2019. lien vers chapitre.

- 3 Benjamin Bergougnoux and Mamadou Moustapha Kanté. Fast exact algorithms for some connectivity problems parametrized by clique-width. *To appear at Theoretical Computer Science*, 2017. URL: <https://hal.archives-ouvertes.fr/hal-01560555>.
- 4 Benjamin Bergougnoux, Mamadou Moustapha Kanté, and O-joung Kwon. An Optimal XP Algorithm for Hamiltonian Cycle on Graphs of Bounded Clique-Width. In *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings*, pages 121–132, 2017. doi:10.1007/978-3-319-62127-2_11.
- 5 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inform. and Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.
- 6 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Boolean-Width of Graphs. In Jianer Chen and Fedor V. Fomin, editors, *IWPEC*, volume 5917 of *Lecture Notes in Computer Science*, pages 61–74. Springer, 2009. doi:10.1007/978-3-642-11269-0_5.
- 7 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theoret. Comput. Sci.*, 511:66–76, 2013. doi:10.1016/j.tcs.2013.01.009.
- 8 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.
- 9 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 (extended abstract). In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science—FOCS 2011*, pages 150–159. IEEE Computer Soc., Los Alamitos, CA, 2011. doi:10.1109/FOCS.2011.23.
- 10 Reinhard Diestel. *Graph Theory*. Number 173 in Graduate Texts in Mathematics. Springer, third edition, 2005.
- 11 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014. doi:10.1137/130910932.
- 12 Robert Ganian and Petr Hliněný. On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width. *Discrete Appl. Math.*, 158(7):851–867, 2010. doi:10.1016/j.dam.2009.10.018.
- 13 Petr A. Golovach, Pinar Heggernes, Mamadou Moustapha Kanté, Dieter Kratsch, Sigve Hortemo Sæther, and Yngve Villanger. Output-Polynomial Enumeration on Graphs of Bounded (Local) Linear MIM-Width. *Algorithmica*, 80(2):714–741, 2018. doi:10.1007/s00453-017-0289-1.
- 14 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. In *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, pages 21:1–21:13, 2017. doi:10.4230/LIPIcs.IPEC.2017.21.
- 15 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. A Unified Polynomial-Time Algorithm for Feedback Vertex Set on Graphs of Bounded Mim-Width. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 42:1–42:14, 2018. doi:10.4230/LIPIcs.STACS.2018.42.
- 16 Ki Hang Kim. *Boolean matrix theory and applications*, volume 70. Dekker, 1982.
- 17 Pedro Montealegre and Ioan Todinca. On Distance-d Independent Set and Other Problems in Graphs with "few" Minimal Separators. In *Graph-Theoretic Concepts in Computer Science - 42nd International Workshop, WG 2016, Istanbul, Turkey, June 22-24, 2016, Revised Selected Papers*, pages 183–194, 2016. doi:10.1007/978-3-662-53536-3_16.
- 18 Sang-Il Oum. *Graphs of Bounded Rank Width*. PhD thesis, Princeton University, 2005.
- 19 Sang-il Oum, Sigve Hortemo Sæther, and Martin Vatshelle. Faster algorithms for vertex partitioning problems parameterized by clique-width. *Theoret. Comput. Sci.*, 535:16–24, 2014. doi:10.1016/j.tcs.2014.03.024.

17:14 Connectivity and Acyclicity Constraints Versus d -Neighbor Equivalence

- 20 Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.
- 21 Michaël Rao. *Décompositions de Graphes et Algorithmes Efficaces*. PhD thesis, Université Paul Verlaine, Metz, 2006.
- 22 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.
- 23 Martin Vatshelle. *New width parameters of graphs*. PhD thesis, University of Bergen, Bergen, Norway, 2012.

Online Bin Covering with Limited Migration

Sebastian Berndt

Department of Computer Science, Kiel University, Kiel, Germany
seb@informatik.uni-kiel.de

Leah Epstein

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

Klaus Jansen

Department of Computer Science, Kiel University, Kiel, Germany
kj@informatik.uni-kiel.de

Asaf Levin

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

Marten Maack

Department of Computer Science, Kiel University, Kiel, Germany
mmaa@informatik.uni-kiel.de

Lars Rohwedder

Department of Computer Science, Kiel University, Kiel, Germany
lro@informatik.uni-kiel.de

Abstract

Semi-online models where decisions may be revoked in a limited way have been studied extensively in the last years.

This is motivated by the fact that the pure online model is often too restrictive to model real-world applications, where some changes might be allowed. A well-studied measure of the amount of decisions that can be revoked is the migration factor β : When an object o of size $s(o)$ arrives, the decisions for objects of total size at most $\beta \cdot s(o)$ may be revoked. Usually β should be a constant. This means that a small object only leads to small changes. This measure has been successfully investigated for different, classical problems such as bin packing or makespan minimization. The dual of makespan minimization – the Santa Claus or machine covering problem – has also been studied, whereas the dual of bin packing – the bin covering problem – has not been looked at from such a perspective.

In this work, we extensively study the bin covering problem with migration in different scenarios. We develop algorithms both for the static case – where only insertions are allowed – and for the dynamic case, where items may also depart. We also develop lower bounds for these scenarios both for amortized migration and for worst-case migration showing that our algorithms have nearly optimal migration factor and asymptotic competitive ratio (up to an arbitrary small ε). We therefore resolve the competitiveness of the bin covering problem with migration.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases online algorithms, dynamic algorithms, competitive ratio, bin covering, migration factor

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.18

Related Version A full version of the paper is available at <https://arxiv.org/abs/1904.06543>.

Acknowledgements This work was partially supported by the DFG Project, “Robuste Online-Algorithmen für Scheduling- und Packungsprobleme”, JA 612 /19-1, and by GIF-Project “Polynomial Migration for Online Scheduling”.



© Sebastian Berndt, Leah Epstein, Klaus Jansen, Asaf Levin, Marten Maack, and Lars Rohwedder; licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 18; pp. 18:1–18:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Online algorithms aim to maintain a competitive solution without knowing future parts of the input. The competitive ratio of such an algorithm (for a maximization problem) is thus defined as the worst-case ratio between the value of an optimal solution produced by an offline algorithm knowing the complete input and the value of the solution produced by the online algorithm. Furthermore, once a decision is made by these algorithms, this decision is fixed and irreversible. While a surprisingly large number of problems do have such algorithms, the complete irreversibility requirement is often too strict, leading to high competitive ratios. Furthermore, if the departure of objects from the instance is also allowed, irreversible online algorithms are rarely able to be competitive at all. From a practical point of view, this is quite alarming, as the departure of objects is part of many applications. We call such a problem *dynamic* and the version with only insertions *static*.

A number of different scenarios to loosen the strict requirement of irreversibility – called *semi-online* scenarios – have been developed over time in order to find algorithms that achieve good competitive ratios for some of the scenarios with bounded reversibility. In the last few years, the concept of the *migration factor* has been studied intensively [3, 10, 11, 12, 13, 16, 19, 21, 22]. Roughly speaking, a migration factor of β allows to reverse a total size of $\beta \cdot s(o)$ decisions, where $s(o)$ denotes the *size* of the newly arrived object o . For a packing problem, this means that the algorithm is allowed to repack objects with a total size of $\beta \cdot s(o)$. This notion of reversibility is very natural, as it guarantees that a small object can only lead to small changes in the solution structure. Furthermore, algorithms with bounded migration factor often show a very clear-cut tradeoff between their migration and the competitive ratio: Many algorithms in this setting have a bounded migration factor which can be defined as a function $f(\varepsilon)$ (growing with $\frac{1}{\varepsilon}$) and a small competitive ratio $g(\varepsilon)$ (growing with ε), where the functions f and g can be defined for all $\varepsilon > 0$ [3, 10, 13, 16, 19, 21, 22]. Such algorithms are called *robust*, as the amount of reversibility allowed only depends on the solution guarantee that one wants to achieve. Such robust algorithms thus serve as evidence for the possibility for sensitivity analysis in approximated settings.

Many different problems have been studied in online and semi-online scenarios, but two problems that have been considered in nearly every scenario are classical scheduling problems: The *bin packing* problem and the *makespan minimization* problem. Both of these problems have been studied intensively in the migration model [3, 10, 12, 13, 19, 21, 22]. Both of these problems also have corresponding dual maximization variants. The dual version of the makespan minimization problem, often called the *Santa Claus* or *machine covering* problem, has also been studied with migration [16, 22]. In contrast, the dual version of bin packing, called *bin covering* has not yet been studied in this model. The aim of this paper is to remedy this situation by taking a look at this classical scheduling problem in the migration model.

Formal Problem Statement. In the *bin covering problem*, a set of items Γ with sizes $s: \Gamma \rightarrow (0, 1]$ is used to cover as many unit sized bins as possible, that is, Γ has to be partitioned maximizing the number of partitions with summed up item size of at least one. An instance of the problem will usually be denoted as I and is given as a sequence of entries $(i, s(i))$ where i is the identifier of the item and $s(i)$ is the size of the item. A *solution* to such an instance I with items Γ is a partition $P: \Gamma \rightarrow \mathbb{N}$ and a set $B = P^{-1}(k)$ with $B \neq \emptyset$ is called a *bin* and we say that the items in B are packed into the k -th bin. A bin B is *covered* if $s(B) := \sum_{i \in B} s(i) \geq 1$, where $s(B)$ is called the load of B , and the goal is to maximize the number of such covered bins. The optimal (maximum) number of covered bins of instance I is denoted as $\text{OPT}(I)$.

We also use the following notation throughout our work: The smallest size of an item in bin B is defined as $s_{\min}(B) := \min_{i \in B} \{s(i)\}$ and the largest size is defined as $s_{\max}(B) := \max_{i \in B} \{s(i)\}$. If \mathcal{B} is a set of bins, we also define its total size $s(\mathcal{B}) := \sum_{B \in \mathcal{B}} s(B)$, its minimal size $s_{\min}(\mathcal{B}) := \min_{B \in \mathcal{B}} s_{\min}(B)$, and its maximal size $s_{\max}(\mathcal{B}) := \max_{B \in \mathcal{B}} s_{\max}(B)$. Furthermore, we define $s_{\min}(\emptyset) = +\infty$ and $s_{\max}(\emptyset) = 0$.

We consider variants of static and dynamic online bin covering in which algorithms are allowed to reassign a bounded amount of previously assigned items. In particular, an algorithm has a *migration factor* of β , if the total size of items that it reassigns upon arrival or departure of an item of size s is bounded by βs . Moreover, it has an *amortized migration factor* of β , if at any time the total size of items that have been reassigned by the algorithm in total is bounded by βS , where S is the total size of all items that arrived before. Intuitively, an item of size s creates a migration potential of size βs upon arrival, and this potential may be used by an algorithm to reassign items right away (non-amortized) or anytime from then on (amortized). Note that if an algorithm has a non-amortized migration factor of β , it also has an amortized migration factor of at most β . Thus, we study four variants in this work.

Offline bin covering is NP-hard and therefore there is little hope for a polynomial time algorithm solving the problem to optimality, and in the online setting there is no algorithm that can maintain an optimal solution regardless of its running time. We prove that this non-existence of algorithms that maintain an optimal solution holds also for (static or dynamic) algorithms with bounded amortized migration factor (and thus also for algorithms with bounded non-amortized migration factor).

Hence, algorithms satisfying some performance guarantee are studied. In particular an offline algorithm ALG for a maximization problem has an *asymptotic performance guarantee* of $\alpha \geq 1$, if $\text{OPT}(I) \leq \alpha \cdot \text{ALG}(I) + c$, where $\text{OPT}(I)$ and $\text{ALG}(I)$ are the objective values of an optimal solution or the one produced by ALG respectively for some instance I , and c is an input independent constant. If $c = 0$ holds, α is called *absolute* rather than asymptotic. An online algorithm has a (asymptotic or absolute) *competitive ratio* of α , if after each arrival or departure an (asymptotic or absolute) performance guarantee of α for the instance of the present items holds. Note that we use the convention of competitive ratios larger than 1 for maximization problems. For minimization problems similar definitions are used but they use the required inequality $\text{ALG}(I) \leq \alpha \cdot \text{OPT}(I) + c$. As we study asymptotic competitive ratios in this paper, we will sometimes omit the word asymptotic (and we always use the word absolute for absolute competitive ratios).

Our Results. We present competitive algorithms using both amortized migration and non-amortized migration and develop nearly matching lower bounds (up to an arbitrarily small additive term of ε). These bounds show the optimality of all of our algorithms for both the static and the dynamic version of the bin covering problem. The main technical contribution of our work is an algorithm with competitive ratio $3/2 + \varepsilon$ and non-amortized migration of $\mathcal{O}(1/\varepsilon^5 \cdot \log^2(1/\varepsilon))$ for the dynamic bin covering problem where items arrive and depart. A major obstacle in the design of competitive algorithms for dynamic problems is the impossibility of moving large items on the arrival or departure of small items. We overcome this obstacle by developing a delicate technique to combine the packing of large and small items. The main results of this work are summarized in the following table. Note that the lower bound of 1 in the third row indicates that there is no online algorithm that maintains an optimal solution with amortized migration factor $\mathcal{O}(1)$. All of our algorithms run in polynomial time. Curiously, we achieve a polynomial migration factor, while most known migration factors are exponential (e.g. for the makespan minimization problem [21]) with the exception of bin packing [3, 18].

Amortization	Departures	Lower Bound	Competitive Ratio	Migration
✘	✘	3/2	$3/2 + \varepsilon$	$\mathcal{O}(1/\varepsilon)$
✘	✓	3/2	$3/2 + \varepsilon$	$\mathcal{O}(1/\varepsilon^5 \cdot \log^2(1/\varepsilon))$
✓	✘	1	$1 + \varepsilon$	$\mathcal{O}(1/\varepsilon)$
✓	✓	3/2	$3/2 + \varepsilon$	$\mathcal{O}(1/\varepsilon^5 \cdot \log^2(1/\varepsilon))$

Due to space constraints, we focus on the non-amortized case with only insertions (described in the first row) and prove most of the corresponding results here. The remaining results are shortly described and a full presentation of them is given in the appendix.

Related Results

Bin Covering. The offline bin covering problem was first studied by Assmann et al. [1]. It was shown that a simple greedy strategy achieves approximation ratio 2. For the online version of the bin covering problem, Csirik and Totik showed in [6] that this simple greedy algorithm also works in the online setting and that the competitive ratio of 2 reached by this algorithm is the best possible. Csirik, Johnson, and Kenyon presented an asymptotic polynomial time approximation scheme (APTAS) with approximation ratio $1 + \varepsilon$ in [5]. This was improved to an asymptotic fully polynomial time approximation scheme (AFPTAS) by Jansen and Solis-Oba in [20]. Many different variants of this problem have also been investigated: If a certain number of classes needs to be part of each bin [9, 15]; if items are drawn probabilistically [14, 15]; if bins have different sizes [7, 23]; if the competitiveness is not measured with regard to an optimal offline algorithm [4, 8]. More variants are e. g. discussed in [17] and lower bounds for several variants are studied in [2].

Makespan Minimization and Santa Claus. The migration factor model was introduced by Sanders, Sivadasan, and Skutella in [21]. The paper investigated several algorithms for the makespan minimization problem and also presents an approximation scheme with absolute competitive ratio $1 + \varepsilon$ and non-amortized migration factor $2^{O(1/\varepsilon \log^2(1/\varepsilon))}$. Skutella and Verschae [22] studied a dynamic setting with amortized migration, where jobs may also depart from the instance. They achieved the same absolute competitive ratio, but their algorithm needs an amortized migration of $2^{O(1/\varepsilon \log^2(1/\varepsilon))}$. Their algorithm also works for the Santa Claus (or machine covering) problem, for which they show that even in the static setting no algorithm has absolute competitive ratio $1 + \varepsilon$ and a bounded migration factor. If one aims for a polynomial migration factor for the Santa Claus problem, Gálvez, Soto, and Verschae presented an online variant of the LPT (longest processing time) algorithm achieving an absolute competitive ratio of $4/3 + \varepsilon$ with non-amortized migration factor $O(1/\varepsilon^3 \log(1/\varepsilon))$ [16]. For the makespan minimization problem with preemption, Epstein and Levin showed in [12] that an optimal algorithm with a non-amortized migration factor of $1 - 1/m$ is achievable and best possible.

Bin Packing. Epstein and Levin presented an approximation scheme with the same ratio $1 + \varepsilon$ and the same non-amortized migration factor $2^{O(1/\varepsilon \log^2(1/\varepsilon))}$ as in the makespan minimization for the bin packing problem in [10]. This result was improved by Jansen and Klein in [18], who drastically reduced the migration factor to $O(1/\varepsilon^4)$. Berndt, Jansen, and Klein used a similar approach to also handle the dynamic bin packing problem, where items may also depart over time [3]. They also showed that a non-amortized migration factor of $\Omega(1/\varepsilon)$ is needed for this. A generalized model, where an item i has arbitrary

movement costs c_i – not necessarily linked to the size of an item – was studied by Feldkord et al. [13]. They showed that for $\alpha \approx 1.387$ and every $\varepsilon > 0$, a competitive ratio of $\alpha + \varepsilon$ is achievable with migration $O(1/\varepsilon^2)$, but no algorithm with migration $o(n)$ and ratio $\alpha - \varepsilon$ exists. Strengthening the lower bound of [3], they also showed an amortized migration factor of $\Omega(1/\varepsilon)$ is needed for the standard model, where movement costs c_i equal items sizes s_i , if one wants to achieve competitive ratio $1 + \varepsilon$. A generalization of bin packing – packing d -dimensional cubic items into unit size cubes – was studied by Epstein and Levin [11].

2 Non-amortized Migration in the Static Case

We begin our study by analyzing the static case with non-amortized migration. We will first present a lower bound showing that no algorithm with constant non-amortized migration factor can have a competitive ratio below $3/2$. Then, we present an algorithm that achieves for all $\varepsilon > 0$ a competitive ratio of $3/2 + \varepsilon$ with non-amortized migration factor $O(1/\varepsilon)$.

We start with a simple lower bound on the asymptotic competitive ratio of all algorithms with constant non-amortized migration factor. This lower bound can also be proved for a different definition of the asymptotic competitive ratio α , where we require $\text{OPT}(I) \leq \alpha \cdot \text{ALG}(I) + o(\text{OPT}(I))$.

► **Proposition 1.** *There is no algorithm for static online bin covering with a constant non-amortized migration factor and an asymptotic competitive ratio smaller than $3/2$.*

Proof sketch. Fix a migration factor β and an integer N . First, insert $6N$ items of size $1 - \varepsilon$ and then $6N$ items of size ε , where $\varepsilon = \min\{(2\beta + 2)^{-1}\}$. ◀

We will now give our algorithm ALG for this scenario. In addition to the instance I , a parameter $\varepsilon > 0$ is also given that regulates the asymptotic competitive ratio and the used migration. The assumption $\varepsilon \leq 0.5$ is justified as for $\varepsilon \geq 0.5$, the result follows by the online algorithm with an asymptotic competitive ratio of 2 presented in [6], or by using the algorithm below with $\varepsilon = \frac{1}{2}$.

► **Theorem 2.** *For each $\varepsilon \in (0, 0.5]$, there is an algorithm ALG for static online bin covering with polynomial running time, an asymptotic competitive ratio of $1.5 + \varepsilon$ with additive constant 3, and a non-amortized migration factor of $\mathcal{O}(1/\varepsilon)$.*

The algorithm distinguishes between big, medium and small items. For each item, it calls a corresponding insertion procedure based on this classification into three classes. An item i is called *big* if $s(i) \in (0.5, 1]$, *medium* if $s(i) \in (\varepsilon, 0.5]$, and *small* otherwise. For a bin B , let $\text{small}(B)$ be the set of small items of B . We define $\text{medium}(B)$ and $\text{big}(B)$ accordingly and also extend these notions to sets of bins \mathcal{B} . Furthermore, we call a covered bin *barely covered*, if removing the biggest item of the smallest class of items, i. e. big, medium or small, contained in the bin, results in the bin not being covered anymore. For instance, consider a bin containing four items with sizes $0.65, 0.3, \varepsilon$ and 0.25ε , for $\varepsilon \geq 0.04$. This bin contains items of all three classes if $\varepsilon < 0.3$. In this case, the biggest item of the smallest class has size ε , and if $\varepsilon = 0.1$, the bin is indeed barely covered. However, if $\varepsilon = 0.22$, the bin is not barely covered. If $\varepsilon = 0.3$, the bin only has items of two classes, and it is not barely covered, since removing one item of size 0.3 results in a total size above 1. Note that showing that removing an arbitrary item of the smallest class of items for a given covered bin results in an uncovered bin is sufficient for showing that it is barely covered.

Let B be a barely covered bin. If B contains at most one big item, its load is bounded from above by 1.5, and if B additionally contains no medium item the bound is reduced to $1 + \varepsilon$. This holds due to the following. Since the big item has size below 1, the bin contains at

least one medium or small item. If the bin has no small items, removing the largest medium item reduces the load to below 1, and together with the medium item the load is below 1.5. If it has a small item, a similar calculation shows that the load is below $1 + \varepsilon$.

The last two types of bins are benign in the sense that they allow analysis using arguments that are based on sizes. This is not the case for bins containing two big items. Such bins could have a size arbitrarily close to 2 (even if they have no other items). Bins of this type are needed for instances with many big items (for an example we refer to the construction of the lower bound in Proposition 1). However, they cause problems not only because they are wastefully packed, but also because they exclusively contain big items that should only be moved if suitably large items arrive in order to bound migration. The basic idea of the algorithm is to balance the number of bins containing two big items and the number of bins containing one big item and no medium items. The two numbers will be roughly the same, which is obtained using migration on arrivals of big and medium items. As described in the previous paragraph, the guarantees of these bins that are based on loads cancel each other out in the sense that an average load not exceeding $1.5 + \varepsilon/2$ can be achieved. In order to keep the number of bins with two big items in check, our algorithm will only produce very few bin types and we will maintain several invariants. This structured approach allows us also to bound the migration needed. We elaborate on the details of the algorithm.

Bin Types and Invariants

We distinguish different types of bins packed by the algorithm.

The bins are partitioned into bins containing two big items (and no other items) **BB**; barely covered bins containing one big item and some medium items **BM**; *barely covered* bins containing one big item and some small items **BSC**; bins that are not covered (*partially covered*) and contain one big item and no medium items (but it could contain small items) **BSP**; and bins that are at most barely covered (they are barely-covered, or not covered) and exclusively contain small or medium items **S** or **M** respectively. Furthermore, let $MC \subseteq M$ and $SC \subseteq S$ be the corresponding subsets of barely covered bins (while $M \setminus MC$ and $S \setminus SC$ are sets of bins that are not covered). We denote the (disjoint) union of **BSC** and **BSP** as **BS**. The set of bins packed by the algorithm (covered or not covered) is denoted as **Bins**. All bins covered by the algorithm are in fact barely covered, and no bin (covered or not) contains items of all three classes. Among bins that are not covered, there are no bins containing a big item and a non-empty set of medium items.

We will now introduce the invariants needed. The first invariant of the algorithm ensures that this bin structure is maintained and the second invariant was already indicated above ($A \dot{\cup} B$ denotes the disjoint union of A and B):

I1 The solution has the proposed bin type structure, i. e. $\text{Bins} = \text{BB} \dot{\cup} \text{BM} \dot{\cup} \text{BS} \dot{\cup} \text{M} \dot{\cup} \text{S}$ and $\text{BS} = \text{BSC} \dot{\cup} \text{BSP}$.

I2 The sets **BB** and **BS** are balanced in size, i. e. $||\text{BB}| - |\text{BS}|| \leq 1$.

Therefore we have $\text{ALG}(I) = |\text{BB}| + |\text{BM}| + |\text{BSC}| + |\text{MC}| + |\text{SC}|$. Furthermore, we have several invariants concerning the distribution of items to different bin types. The intuition behind these invariants is always the same: We have to ensure that no other algorithm is able to use the small and medium items to cover too many bins.

I3 The big items contained in **BM** are at least as big as the ones in **BSC** which in turn are at least as big as the ones in **BSP**, and the smallest big items are placed in **BB**, i. e. $s(i) \geq s(i')$ for each (i) $i \in \text{big}(\text{BM})$ and $i' \in \text{big}(\text{BS} \cup \text{BB})$; each (ii) $i \in \text{big}(\text{BM} \cup \text{BSC})$ and $i' \in \text{big}(\text{BSP} \cup \text{BB})$; each (iii) $i \in \text{big}(\text{BM} \cup \text{BS})$ and $i' \in \text{big}(\text{BB})$. Informally, $\text{BB} \leq \text{BSP} \leq \text{BSC} \leq \text{BM}$.

I4 The items in M cannot be used to cover a bin together with a big item from BS or BB , i. e. $BS \cup BB \neq \emptyset \implies s(M) < 1 - s_{\max}(BS \cup BB)$.

I5 If a bin containing only small items exists, all bins in BS are covered, i. e. $|S| > 0 \implies |BSP| = 0$.

Lastly, there are some bin types with bins that are not covered, and we have to make sure that they are not wastefully packed:

I6 If there are small items in BSP , they are all included in the bin containing the biggest item in BSP .

I7 Both S and M each contain at most one bin that is not covered.

This concludes the definition of all invariants. It is easy to see that the invariants all hold in the beginning when no item has arrived yet. Next, we describe the insertion procedures and argue that the invariants are maintained.

Insertion Procedures

We start with the definition of two simple auxiliary procedures used in the following:

- **GreedyPush**(i, \mathcal{B}) is given an item i and a set of bins \mathcal{B} . If all the bins contained in \mathcal{B} are covered, it creates a new bin containing item i , and otherwise it inserts i into the most loaded bin that is not covered.
- **GreedyPull**(B, \mathcal{B}) is given bin B and a set of bins \mathcal{B} . It successively removes a largest non-big item from a least loaded bin from $\{B' \in \mathcal{B} \mid \text{small}(B') \cup \text{medium}(B') \neq \emptyset\}$ and inserts it into B . This is repeated until B is covered or \mathcal{B} does not contain non-big items.

Consider one application of **GreedyPull** such that B already has a big item. The total size of moved items is smaller than 1. Both procedures are used to insert and repack non-big items. Note that calling **GreedyPush** for a small item and bin set BSP or S , or a medium item and bin set M , the last two invariants **I6** and **I7** are maintained. For BSP , the most loaded bin always contains the largest big item, and if there is at least one small item, such a bin is unique. It could happen that as a result of inserting a small item into this bin of BSP the bin is covered and moves to BSC .

For each insertion procedure, we will argue that the invariants are maintained and focus on the critical ones, that is, in each context the invariants, that are not discussed explicitly, trivially hold. For example, we do not discuss **I1** in the following, because it will always be easy to see that it is maintained.

Insertion of Small Items. If the arriving item i^* is small, we call **GreedyPush**(i^*, BSP), if $BSP \neq \emptyset$, and **GreedyPush**(i^*, S) otherwise. Insertion into a bin of BSP (the most loaded one) may lead to a covered bin, in which case the bin becomes a bin of BSC (but remains in BS). It is easy to verify, that all invariants, and **I5**, **I6** and **I7** in particular, are maintained by this. As mentioned above, **I3** holds, as the most loaded bin in BSP always contains the largest big item. Furthermore, there is no migration in this case. The insertion of a medium or big item, however, is more complicated.

Insertion of Big Items. In the case that a big item i^* arrives, we have to be careful where we place it exactly, because, on the one hand, the distributions of big and medium items, that is, invariants **I3** and **I4**, have to be maintained, and, on the other hand, we have to balance out BS and BB (**I2**). We consider placing the item in BM , BS or BB in this order, i. e. we first try to insert i^* into BM , then into BS and finally into BB . Figure 1 illustrates this.

Insertion into BM. We insert i^* into BM, if either $s(i^*) + s(\mathbf{M}) \geq 1$ or $s(i^*) > s_{\min}(\text{big}(\mathbf{BM}))$. Note that the first condition implies $s(i^*) \geq s_{\max}(\text{big}(\mathbf{BB} \cup \mathbf{BS}))$, because of I4, and therefore the insertion of i^* into BM maintains I3 in both situations. The second condition implies $\mathbf{BM} \neq \emptyset$, because we set $s_{\min}(\emptyset) = +\infty$. In either of these cases, we create a new bin $B^* = \{i^*\}$ and call $\text{GreedyPull}(B^*, \mathbf{M})$, thereby ensuring that I4 is maintained if the new bin is covered. If the first condition did hold, B^* is covered afterwards and we do nothing else. Otherwise, there is a bin $B \in \mathbf{BM}$ containing a big item i with $s(i) = s_{\min}(\text{big}(\mathbf{BM})) < s(i^*)$, and we have $\mathbf{M} = \emptyset$. We remove i from B , yielding $\mathbf{M} = \{B\}$, and call $\text{GreedyPull}(B^*, \mathbf{M})$ a second time. Afterwards, B^* is covered, because $s(i^*) > s(i)$. Furthermore, $s(i) + s(\mathbf{M}) < 1$, because B was barely covered before and the biggest medium item was removed from B due to the second call of GreedyPull . This ensures I4, since by I3, item i is no smaller than any big item packed in BS or BB. The item i is reinserted using a recursive call to the procedure of inserting a big item. However, item i will not be considered for insertion into BM, because neither the first nor second condition holds for this item, and the other insertion options have no recursive calls for insertion into BM. It is easy to verify that the distribution of medium items in \mathbf{M} (I7) is maintained.

Insertion into BS. This step is possible only for item i^* that satisfies $s(i^*) + s(\mathbf{M}) < 1$ and $s(i^*) \leq s_{\min}(\text{big}(\mathbf{BM}))$. Thus, \mathbf{BM} will have the largest big items as required in Invariant I3 after the insertion is performed. In this case there is no recursive call for inserting a big item.

We insert i^* into BS, if either $s(i^*) > s_{\min}(\text{big}(\mathbf{BS}))$ or the following two conditions hold: $s(i^*) \geq s_{\max}(\text{big}(\mathbf{BB}))$ and $|\mathbf{BS}| \leq |\mathbf{BB}|$. Note that $s(i^*) \geq s_{\max}(\text{big}(\mathbf{BB}))$ trivially holds, if $\mathbf{BB} = \emptyset$. Inserting i^* into BS under these conditions already ensures the correct distribution of big items (I3) with respect to BS and BB, but we still have to be careful concerning the distribution within the two subsets of BS. The procedure is divided into three simple steps. As a first step, we create a new bin $B^* = \{i^*\}$ and call $\text{GreedyPull}(B^*, \mathbf{S})$. No matter whether B^* is now covered or not, Invariant I5 is satisfied as either B^* is covered and therefore $\mathbf{BSP} = \emptyset$ both before and after the call, or B^* is not covered but now $\mathbf{S} = \emptyset$ (it is possible that both will hold). Note that all properties of the invariants are satisfied, if B^* is already covered. In particular, Invariant I3 holds within BS because $\mathbf{BSP} = \emptyset$. In the remainder of the second step of the insertion into BS algorithm we deal with the case that B^* is not covered.

Let \mathbf{XB} denote the set of bins $B \in \mathbf{BS}$ that include small items as well as a big item i with $s(i) < s(i^*)$. Recall that any bin of BSC has at least one small item, while at most one bin of BSP has small items.

First, assume that $\mathbf{XB} = \emptyset$ but B^* is not covered. There are two cases. In the first case, at least one item of \mathbf{S} was moved. In this case before we started dealing with i^* , the set \mathbf{BSP} was empty, and now B^* is the unique bin of \mathbf{BSP} , and its big item is not larger than those of BSC (if the last set is not empty) so the invariants I3 and I6 are maintained. In the second case, \mathbf{S} was empty, and B^* is now a bin of \mathbf{BSP} with only a big item. Since $\mathbf{XB} = \emptyset$, adding B^* to \mathbf{BSP} maintains the invariants I3 and I6. Thus, it is left to deal with the case $\mathbf{XB} \neq \emptyset$. In the remainder of the second step of the insertion into BS algorithm we deal with the case that \mathbf{XB} is not empty.

As B^* is not yet covered, we now have $\mathbf{S} = \emptyset$, and this might have been the case before the call of GreedyPull , in particular if we had $\mathbf{BSP} \neq \emptyset$. Due to the existence of a big item that is smaller than i^* in BS (such items exist in all bins of \mathbf{XB}), we have to be careful in order to maintain the correct distribution of big and small items inside of BS (I3 and I6).

In the second step, we construct a set of bins $\tilde{\mathbf{B}} \subseteq \mathbf{BS}$ from which small items are removed in order to cover B^* . If $\mathbf{BSP} \cap \mathbf{XB} \neq \emptyset$, this set has exactly one bin (containing small items) by Invariant I6. If such a bin exists, we denote it by B_1 . If $\mathbf{BSC} \cap \mathbf{XB} \neq \emptyset$, the set BSC includes

a bin that contains a big item i' with $s_{\min}(\text{big}(\text{BSC})) = s(i') < s(i^*)$ and we denote one such bin (with a big item of minimum size in BSC) by B_2 . As $\text{XB} \neq \emptyset$, at least one of the bins B_1 or B_2 must exist, but it can also be the case that both exist. Let $\tilde{\text{B}}$ be the set of cardinality 1 or 2, which contains these bins. The next operation of the second step is to call $\text{GreedyPull}(B^*, \tilde{\text{B}})$. It is easy to see that no matter whether B^* is covered or not after this operation, the invariants I3 and I6 hold. Specifically, if B_2 does not exist, B^* is not necessarily covered, but I3 and I6 hold as all big items of BSC are not smaller than i^* (as every such bin has at least one small item). If B_2 exists, then B^* keeps receiving items coming first from B_1 and then possibly also from B_2 , until it is covered. As the total size of small items of B_2 is sufficient for covering B^* since the big item of B_2 is smaller than i^* , B^* will be covered, so all big items of BSP are not larger than i^* .

Lastly, we describe the third step, which is performed for all cases above, after i^* has been inserted. The insertion of i^* might have violated I2, that is, we now have $|\text{BS}| = |\text{BB}| + 2$. In this case, we perform the last step, namely, we select two bins $B_3, B_4 \in \text{BS}$ with minimal big items, merge the big items into a BB bin and remove and reinsert all small items from B_3 and B_4 , using insertion of small items. This yields, $|\text{BS}| = |\text{BB}| - 1$ and I2 holds.

Insertion into BB. If i^* was not inserted in any of the last steps, it is inserted into BB. In this case, we know from the conditions above that $s(i^*) \leq s_{\min}(\text{big}(\text{BS} \cup \text{BM}))$, and additionally that $s(i^*) \geq s_{\max}(\text{big}(\text{BB}))$ implies $|\text{BS}| = |\text{BB}| + 1$. We consider two cases.

If $|\text{BS}| = |\text{BB}| + 1$ (and hence $\text{BS} \neq \emptyset$), we select a bin $B \in \text{BS}$ with a big item of minimal size. We insert i^* into B to obtain a BB bin and remove and reinsert all small items from B . This yields $|\text{BS}| = |\text{BB}| - 1$ and I2 holds.

If $|\text{BS}| < |\text{BB}| + 1$, we have $s(i^*) < s_{\max}(\text{big}(\text{BB}))$ (and hence $\text{BB} \neq \emptyset$). In this case, we select a bin $B \in \text{BB}$ with a big item i of maximal size, insert i^* into B , and remove and reinsert i . Because of its size and invariant I4, the item i will be inserted into BS. Note that in both cases invariant I3 is maintained.

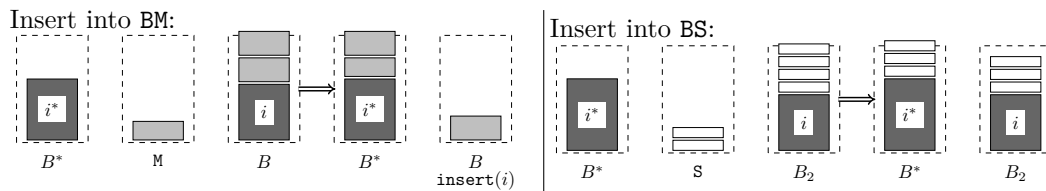


Figure 1 Insertion of a big item i^* . Big items are drawn in dark gray, medium items in light gray, and small items in white.

Insert into BM: open a new bin B^* for i^* ; pull M into B^* ; pull from bin $B \in \text{BM}$ containing the smallest big item i ; remove and reinsert i .

Insert into BS: open a new bin B^* for i^* ; pull S into B^* ; pull from bin $B \in \text{BS}$ containing the smallest big item i .

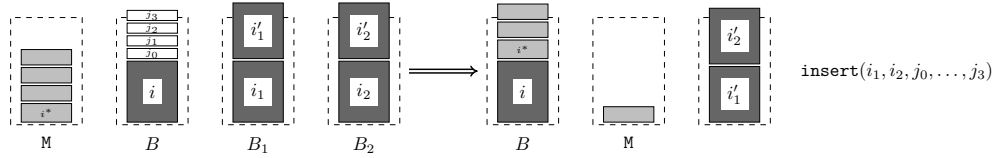
► **Lemma 3.** *The overall size of items migrated due to the insertion of a big item i^* is upper bounded by 11.*

Proof sketch. First, note that an insertion into BS can not trigger the reinsertion of a big item. The insertion into BB can only trigger the reinsertion of a single big item into BS and the insertion into BM can only trigger the reinsertion into BB or BS. Hence, each insertion of a big item can trigger at most two other insertions in total and thus only move a total size of 2 this way. The direct reassignments are bounded by 5. ◀

Insertion of Medium Items. If a medium item i^* arrives, $\text{GreedyPush}(i^*, M)$ is called. Afterwards, the invariant I4 may be infringed and if this happens, we have $BS \cup BB \neq \emptyset$ and $s(M) \geq 1 - s_{\max}(BS \cup BB)$, and we continue as follows. We will now describe how to pack a barely covered bin using the items from M and a largest big item from $BS \cup BB$ to maintain I4.

If $BS = \emptyset$, I2 implies that BB contains a single bin B including two items i and i' with $s_{\max}(BS \cup BB) = s(i) \geq s(i')$. We remove i' from B , and call $\text{GreedyPull}(B, M)$ to create a BM bin. Afterwards, $s(M)$ and $s_{\max}(BS \cup BB)$ are at most as big as they were before i^* arrived as the first item we pulled from M is at least as big as i^* , and therefore I4 holds. Furthermore $BS = BB = \emptyset$ and I2 still holds. Lastly, we reinsert the big item i' .

If, on the other hand, $BS \neq \emptyset$, the corresponding big item i with $s_{\max}(BS \cup BB) = s(i)$ is contained in a bin $B \in BS$, because of I3. In this case, we remove the small items from B and call $\text{GreedyPull}(B, M)$. Afterwards I4 holds, but I2 may be infringed due to the removal of a bin from BS , i.e. $|BB| = |BS| + 2$. In this case, we remove the two biggest items i_1 and i_2 from the bins $B_1, B_2 \in BB$ and if $B_1 \neq B_2$ merge the two bins. This yields $|BB| = |BS| + 1$ and I2 holds. Afterwards, we reinsert the two items i_1 and i_2 , which both will be inserted in BS due to their sizes. No matter whether we had to rebalance $|BB|$ and $|BS|$ or not, we reinsert the removed small items from B as a last step. Figure 2 contains an illustration of this process.



■ **Figure 2** Insertion of a medium item i^* . Big items are drawn in dark gray, medium items in light gray, and small items in white.

► **Lemma 4.** *The overall size of items migrated due to the insertion of a medium item i^* is upper bounded by 27.*

Analysis. The migration bound stated in Theorem 2 or more precisely $\frac{27}{\varepsilon}$ is already implied by Lemma 3 and Lemma 4, as a medium item has size above ε . It is easy to see that:

► **Remark 5.** The presented algorithm for static bin covering has polynomial running time.

Hence, the only thing left to show is the stated asymptotic competitive ratio:

► **Lemma 6.** *The presented algorithm has an asymptotic competitive ratio of $1.5 + \varepsilon$ with additive constant 3.*

Proof. First, we consider the case $BSP = \emptyset$. In this case, the claim holds because the bins on average have not too much excess size. More precisely, we obviously have $\text{OPT}(I) \leq s(I)$, and invariants I1 and I7 imply $s(I) < 2|BB| + (1 + \varepsilon)|BS| + 1.5|BM| + 1.5|MC| + (1 + \varepsilon)|SC| + 2$. Furthermore, we have $0.5|BB| \leq 0.5(|BS| + 1)$, due to Invariant I2, and $|BS| = |BSC|$, as $BSP = \emptyset$ holds in the case we are currently considering. Hence $\text{OPT}(I) < (1.5 + \varepsilon)(|BB| + |BSC| + |BM| + |MC| + |SC|) + 2.5 < (1.5 + \varepsilon) \text{ALG}(I) + 3$.

A similar argument holds, if $BSP \neq \emptyset$ but $BB = \emptyset$. In this case, we have $|BSP| = |BS| = 1$, because of invariant I2; and $S = \emptyset$, because of Invariant I5. Hence $\text{OPT}(I) \leq s(I) < 1.5|BM| + 1.5|MC| + 2 = 1.5 \text{ALG}(I) + 2$.

Next, we consider the case $\text{BSP} \neq \emptyset$ and $\text{BB} \neq \emptyset$. Here, we have $\text{MC} = \emptyset$, because of Invariant I4, and $\text{S} = \emptyset$, because of Invariant I5. Note that every bin of $\text{BSC} \cup \text{BM}$ has at least one item that is not big, since big items have sizes below 1, and these bins are covered. Let $\xi = s_{\max}(\text{BSP})$ be the size of a big item from BSP with maximal size. Then all items in $\text{BB} \cup \text{BSP}$ have size at most ξ (I3) and $\xi > 0.5$. We construct a modified instance I^* as follows:

1. The size of each big item with size below ξ is increased to ξ .
2. Every big item of size larger than ξ is split into a big item of size ξ and a medium or small item, such that the total size of these two items is equal to the size of the original item. Let X be the set of items with sizes of ξ , which we will call ξ -items in the instance after these transformations (X includes also items whose sizes were ξ in I).
3. For each bin from $\text{BSC} \cup \text{BM}$, select the largest item of I that is not big and call it *special*. By increasing item sizes if necessary, change the sizes of all special items to 0.5. Let Y be the set of special items (whose sizes are now all equal to 0.5). Let Z be the set of the remaining items not belonging to X or Y (in the instance I^* after the transformations, so there may be items that did not exist in I resulting from splitting a big item).

The set of items in I^* is just $X \cup Y \cup Z$. For instance I^* , any bin of BB contains only two items of X . Any bin of BSP has an item of X , and one of these bins may also have small items of Z , but it is not covered. Any bin of $\text{BSC} \cup \text{BM}$ has one item of X , one item of Y , and possibly items of Z . There may be one uncovered bin of M , containing items of Z .

Note that $\text{OPT}(I) \leq \text{OPT}(I^*)$, since any packing for I can be used as a packing for I^* with at least the same number of covered bins. Next, we investigate the relationship between $\text{OPT}(I^*)$ and the packing of the algorithm for the original instance I . For some optimal solution for I^* without overpacked bins (more than barely covered), let k_2 , k_1 and k_0 be the number of covered bins with 2, 1 and 0 items from $X \cup Y$, respectively. Then we have $\text{OPT}(I^*) = k_2 + k_1 + k_0$ and due to counting $2k_2 + k_1 = |X \cup Y| = (2|\text{BB}| + |\text{BS}| + |\text{BM}|) + (|\text{BM}| + |\text{BSC}|)$. Since each item in $X \cup Y$ is upper bounded by ξ , we have: $(1 - \xi)k_1 + k_0 \leq s(Z)$.

The total size of items (of Z only) packed into the bin of M is below $1 - \xi$ since BSP has a big item of size ξ in I and by Invariant I4, since every item of $\text{BS} \cup \text{BB}$ is smaller than $1 - s(\text{M})$. For BSP only one bin may contain items of Z by Invariant I6, and this bin has an item of size ξ in I (and it is not covered), so it also has items of Z of total size below $1 - \xi$. Consider a bin of $\text{BSC} \cup \text{BM}$. The total size of items excluding the special item is the same for I and I^* . Since such a bin is barely covered and for I it has items of one class except for the big item (small or medium), removing the special item results in a load below 1. The total size of items of I^* in such a bin excluding the ξ -item and the special item is below $1 - \xi$.

Therefore, we find that $s(Z) \leq (1 - \xi)(|\text{BM}| + |\text{BSC}| + 2)$. Hence:

$$\begin{aligned} 2 \text{OPT}(I) &\leq 2(k_2 + k_1 + k_0) \leq (2k_2 + k_1) + (k_1 + (1 - \xi)^{-1}k_0) \\ &\leq (2|\text{BB}| + |\text{BS}| + |\text{BM}| + |\text{BM}| + |\text{BSC}|) + (|\text{BM}| + |\text{BSC}| + 2) \\ &\stackrel{\text{Invariant I2}}{\leq} 3|\text{BB}| + 3|\text{BM}| + 2|\text{BSC}| + 3 \leq 3 \text{ALG}(I) + 3. \end{aligned} \quad \blacktriangleleft$$

Extending Our Results

Non-amortized Migration in the Dynamic Case. We are able to extend the result of the static case and show that we can also handle the case of departing items.

► **Theorem 7.** *For each $\varepsilon < 1$ with $1/\varepsilon \in \mathbb{Z}$, there is an algorithm ALG for dynamic online bin covering with polynomial running time, an asymptotic competitive ratio of $1.5 + \varepsilon$ with additive constant $\mathcal{O}(\log 1/\varepsilon)$, and a non-amortized migration factor of $\mathcal{O}((1/\varepsilon)^5 \log^2(1/\varepsilon))$.*

18:12 Online Bin Covering with Limited Migration

This is the most elaborate result of the paper and its proof can be found in the long version of the paper (see appendix). In the following, we briefly discuss this the result and give some intuition for the developed techniques.

The main challenge in the dynamic case arises from small items: Let N be some positive integer. Consider the case that N^2 items of size $1/N$ arrived and were placed into N bins, covering each of them perfectly. Next, one item from each bin leaves yielding a solution without any covered bin while the optimum is still $N - 1$. Hence, a migration strategy for the small items is needed. Now, coming up with such a strategy to deal with the present example is rather simple, since all the items are of the same size, but in principle small items may differ in size by arbitrary factors. Still, the case with only small items can be dealt with adapting a technique for online bin packing with migration [3]. The basic idea is to sort the items non-increasingly and maintain a packing that corresponds to a partition of this sequence into barely covered bins. If an item arrives, it is inserted into the correct bin and excess items are pushed to the right, that is, to the neighboring bin containing the next items in the ordering, and this process is repeated until the packing is restored. Correspondingly, if an item departs, items are pulled in from the next bin to the right. In this process the arrival or departure of a small item can only cause movements of items at most as big as the original one. While this is useful, it does not suffice to bound migration: Too many bins have to be repacked. In order to deal with this, the bins are partitioned into *chains* of appropriate constant length with a *buffer bin* at the end, which is used to interrupt the migration process. This technique can be applied for the bins \mathbf{S} containing only small items, but for bins \mathbf{BS} containing big items as well problems arise. The main reason for this is that in order to adapt our analysis, we need to cover the bins in \mathbf{BS} containing larger big items with higher priority and furthermore guarantee that there are no (or only few) bins contained in \mathbf{S} if there are bins containing big items that are not covered, i.e., $\mathbf{BSP} \neq \emptyset$. It is not hard to see that spreading one sequence of chains out over the bins of \mathbf{BS} and \mathbf{S} will not suffice.

To overcome these problems, we develop a novel technique: We partition the bins of \mathbf{BS} into few, that is, $\mathcal{O}(\log 1/\varepsilon)$ many, *groups*. Each of the groups is in turn partitioned into *parallel chains* of length $\mathcal{O}(1/\varepsilon)$. The groups are defined such that they comply with a non-increasing ordering of both the big and the small items: the first group contains the largest big and small items, the next group the remaining largest, and so on. A similar ordering holds for each single parallel chain, but no such structure is maintained in between the parallel chains of the same group. Now, whenever a buffer bin of a parallel chain becomes overfilled, items are pushed directly into the next group. However, to maintain the described structure, these have to be the smallest items of the group, and to guarantee this we introduce some additional structure for the buffer bins in each group. While there may be a chain reaction caused by such a push or pull, the migration can still be bounded, because there are only few groups. We are able to guarantee that there is at most one group \mathbf{G} containing uncovered bins and that that all bins of \mathbf{BS} are barely covered, if $\mathbf{S} \neq \emptyset$. These are the essential properties we need in order to adapt our approach to the dynamic case.

Amortized Migration. First of all, we can strengthen the lower bound of Theorem 1 if items can depart and show that the same bound also holds for amortized migration in this case.

► **Proposition 8.** *There is no algorithm for dynamic online bin covering with a constant amortized migration factor β and an asymptotic competitive ratio smaller than $3/2$.*

If items never depart, we can use the amortization by repacking the completely instance every once in a while with the help of an AFPTAS of Jansen and Solis-Oba [20]. We can also show that we need to be contended with a non-optimal solution by making use of a highly non-trivial construction.

► **Theorem 9.** *For every $\varepsilon > 0$, there is an algorithm for static bin covering with polynomial running time, asymptotic competitive ratio $1 + \varepsilon$, and amortized migration factor $\mathcal{O}(1/\varepsilon)$. Additionally, there is no (possibly exponential time) algorithm for static online bin covering that maintains an optimal solution with constant amortized migration factor β .*

References

- 1 Susan F. Assmann, David S. Johnson, Daniel J. Kleitman, and Joseph Y.-T. Leung. On a Dual Version of the One-Dimensional Bin Packing Problem. *J. Algorithms*, 5(4):502–525, 1984. doi:10.1016/0196-6774(84)90004-X.
- 2 János Balogh, Leah Epstein, and Asaf Levin. Lower bounds for online bin covering-type problems. *Journal of Scheduling*, pages 1–11, 2018.
- 3 Sebastian Berndt, Klaus Jansen, and Kim-Manuel Klein. Fully dynamic bin packing revisited. *Mathematical Programming*, 2018. doi:10.1007/s10107-018-1325-x.
- 4 Marie G. Christ, Lene M. Favrholdt, and Kim S. Larsen. Online bin covering: Expectations vs. guarantees. *Theor. Comput. Sci.*, 556:71–84, 2014. doi:10.1016/j.tcs.2014.06.029.
- 5 János Csirik, David S. Johnson, and Claire Kenyon. Better approximation algorithms for bin covering. In S. Rao Kosaraju, editor, *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA.*, pages 557–566. ACM/SIAM, 2001. URL: <http://dl.acm.org/citation.cfm?id=365411.365533>.
- 6 János Csirik and V. Totik. Online algorithms for a dual version of bin packing. *Discrete Applied Mathematics*, 21(2):163–167, 1988. doi:10.1016/0166-218X(88)90052-2.
- 7 Leah Epstein. Online Variable Sized Covering. *Inf. Comput.*, 171(2):294–305, 2001. doi:10.1006/inco.2001.3087.
- 8 Leah Epstein, Lene M. Favrholdt, and Jens S. Kohrt. Comparing online algorithms for bin packing problems. *J. Scheduling*, 15(1):13–21, 2012. doi:10.1007/s10951-009-0129-5.
- 9 Leah Epstein, Csanád Imreh, and Asaf Levin. Class Constrained Bin Covering. *Theory Comput. Syst.*, 46(2):246–260, 2010. doi:10.1007/s00224-008-9129-7.
- 10 Leah Epstein and Asaf Levin. A robust APTAS for the classical bin packing problem. *Math. Program.*, 119(1):33–49, 2009. doi:10.1007/s10107-007-0200-y.
- 11 Leah Epstein and Asaf Levin. Robust Approximation Schemes for Cube Packing. *SIAM Journal on Optimization*, 23(2):1310–1343, 2013. doi:10.1137/11082782X.
- 12 Leah Epstein and Asaf Levin. Robust Algorithms for Preemptive Scheduling. *Algorithmica*, 69(1):26–57, 2014. doi:10.1007/s00453-012-9718-3.
- 13 Björn Feldkord, Matthias Feldotto, Anupam Gupta, Guru Guruganesh, Amit Kumar, Sören Riechers, and David Wajc. Fully-Dynamic Bin Packing with Little Repacking. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPICs*, pages 51:1–51:24. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPICs.ICALP.2018.51.
- 14 Carsten Fischer and Heiko Röglin. Probabilistic Analysis of the Dual Next-Fit Algorithm for Bin Covering. In Evangelos Kranakis, Gonzalo Navarro, and Edgar Chávez, editors, *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, volume 9644 of *Lecture Notes in Computer Science*, pages 469–482. Springer, 2016. doi:10.1007/978-3-662-49529-2_35.
- 15 Carsten Fischer and Heiko Röglin. Probabilistic Analysis of Online (Class-Constrained) Bin Packing and Bin Covering. In Michael A. Bender, Martin Farach-Colton, and Miguel A. Mosteiro, editors, *LATIN 2018: Theoretical Informatics - 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings*, volume 10807 of *Lecture Notes in Computer Science*, pages 461–474. Springer, 2018. doi:10.1007/978-3-319-77404-6_34.

- 16 Waldo Gálvez, José A. Soto, and José Verschae. Symmetry Exploitation for Online Machine Covering with Bounded Migration. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPICs*, pages 32:1–32:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPICs.ESA.2018.32.
- 17 Teofilo F. Gonzalez, editor. *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, 2007. doi:10.1201/9781420010749.
- 18 Klaus Jansen and Kim-Manuel Klein. A Robust AFPTAS for Online Bin Packing with Polynomial Migration,. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 589–600. Springer, 2013. doi:10.1007/978-3-642-39206-1_50.
- 19 Klaus Jansen, Kim-Manuel Klein, Maria Kosche, and Leon Ladewig. Online Strip Packing with Polynomial Migration. In Klaus Jansen, José D. P. Rolim, David Williamson, and Santosh Srinivas Vempala, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, volume 81 of *LIPICs*, pages 13:1–13:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPICs.APPROX-RANDOM.2017.13.
- 20 Klaus Jansen and Roberto Solis-Oba. An asymptotic fully polynomial time approximation scheme for bin covering. *Theor. Comput. Sci.*, 306(1-3):543–551, 2003. doi:10.1016/S0304-3975(03)00363-3.
- 21 Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online Scheduling with Bounded Migration. *Math. Oper. Res.*, 34(2):481–498, 2009. doi:10.1287/moor.1090.0381.
- 22 Martin Skutella and José Verschae. Robust Polynomial-Time Approximation Schemes for Parallel Machine Scheduling with Job Arrivals and Departures. *Math. Oper. Res.*, 41(3):991–1021, 2016. doi:10.1287/moor.2015.0765.
- 23 Gerhard J. Woeginger and Guochuan Zhang. Optimal on-line algorithms for variable-sized bin covering. *Oper. Res. Lett.*, 25(1):47–50, 1999. doi:10.1016/S0167-6377(99)00023-1.

Computing k -Modal Embeddings of Planar Digraphs

Juan José Besa 

University of California, Irvine, USA
<http://sites.uci.edu/juanbesa>
jjbesavi@uci.edu

Giordano Da Lozzo¹ 

Roma Tre University, Rome, Italy
<http://www.dia.uniroma3.it/~dalozzo>
giordano.dalozzo@uniroma3.it

Michael T. Goodrich 

University of California, Irvine, USA
<https://www.ics.uci.edu/~goodrich>
goodrich@uci.edu

Abstract

Given a planar digraph G and a positive even integer k , an embedding of G in the plane is k -modal, if every vertex of G is incident to at most k pairs of consecutive edges with opposite orientations, i.e., the incoming and the outgoing edges at each vertex are grouped by the embedding into at most k sets of consecutive edges with the same orientation. In this paper, we study the k -MODALITY problem, which asks for the existence of a k -modal embedding of a planar digraph. This combinatorial problem is at the very core of a variety of constrained embedding questions for planar digraphs and flat clustered networks.

First, since the 2-MODALITY problem can be easily solved in linear time, we consider the general k -MODALITY problem for any value of $k > 2$ and show that the problem is NP-complete for planar digraphs of maximum degree $\Delta \geq k+3$. We relate its computational complexity to that of two notions of planarity for flat clustered networks: Planar Intersection-Link and Planar NodeTrix representations. This allows us to answer in the strongest possible way an open question by Di Giacomo et al. [GD17], concerning the complexity of constructing planar NodeTrix representations of flat clustered networks with small clusters, and to address a research question by Angelini et al. [JGAA17], concerning intersection-link representations based on geometric objects that determine complex arrangements. On the positive side, we provide a simple FPT algorithm for partial 2-trees of arbitrary degree, whose running time is exponential in k and linear in the input size. Second, motivated by the recently-introduced planar L-drawings of planar digraphs [GD17], which require the computation of a 4-modal embedding, we focus our attention on $k = 4$. On the algorithmic side, we show a complexity dichotomy for the 4-MODALITY problem with respect to Δ , by providing a linear-time algorithm for planar digraphs with $\Delta \leq 6$. This algorithmic result is based on decomposing the input digraph into its blocks via BC-trees and each of these blocks into its triconnected components via SPQR-trees. In particular, we are able to show that the constraints imposed on the embedding by the rigid triconnected components can be tackled by means of a small set of reduction rules and discover that the algorithmic core of the problem lies in special instances of NAESAT, which we prove to be always NAE-satisfiable – a result of independent interest that improves on Porschen et al. [SAT03]. Finally, on the combinatorial side, we consider outerplanar digraphs and show that any such a digraph always admits a k -modal embedding with $k = 4$ and that this value of k is best possible for the digraphs in this family.

2012 ACM Subject Classification Human-centered computing → Graph drawings; Mathematics of computing → Graph algorithms; Information systems → Data structures

Keywords and phrases Modal Embeddings, Planarity, Directed Graphs, SPQR trees, NAESAT

¹ Corresponding author



Digital Object Identifier 10.4230/LIPIcs.ESA.2019.19

Related Version Due to space limitations, some theorems and proofs are omitted or sketched and can be found in the full version of the paper [21], which is available at <https://arxiv.org/abs/1907.01630>.

Funding This work was supported in part by NSF grant 1815073, by MIUR Project “MODE” under PRIN 20157EFM5C, by MIUR Project “AHeAD” under PRIN 20174LF3T8, by MIUR-DAAD JMP N° 34120, and by H2020-MSCA-RISE project 734922 – “CONNECT”.

1 Introduction

Computing k -modal embeddings of planar digraphs, for some positive even integer k called *modality*, is an important algorithmic task at the basis of several types of graph visualizations. In *2-modal embeddings*, also called *bimodal embeddings*, the outgoing and the incoming edges at each vertex form two disjoint sequences. Bimodal embeddings are ubiquitous in Graph Drawing. For instance, level planar drawings [15, 22] and upward-planar drawings [8, 17] – two of the most deeply-studied graph drawing standards – determine bimodal embeddings. *4-modal embeddings*, where the outgoing and the incoming edges at each vertex form up to four disjoint sequences with alternating orientations, arise in the context of planar L-drawings of digraphs. In an *L-drawing* of an n -vertex digraph, introduced by Angelini et al. [1], vertices are placed on the $n \times n$ grid so that each vertex is assigned a unique x -coordinate and a unique y -coordinate and each edge uv (directed from u to v) is represented as a 1-bend orthogonal polyline composed of a vertical segment incident to u and of a horizontal segment incident to v . Recently, Chaplick et al. [13] addressed the question of deciding the existence of *planar L-drawings*, i.e., L-drawings whose edges might possibly overlap but do not cross and observe that the existence of a 4-modal embedding is a necessary condition for a digraph to admit such a representation (Fig. 1a).

To the best of our knowledge, no further relationships have been explicitly pointed out in the literature between modal embeddings and notable drawing models for modality values greater than four, yet they do exist. Da Lozzo et al. [14] and Di Giacomo et al. [18] study the planarity of *NodeTrix representations* of flat clustered networks, a hybrid representational model introduced by Henry, Fekete, and McGuffin [19], where clusters and intra-cluster edges are represented as adjacency-matrices, with rows and columns for the vertices of each cluster, and inter-cluster edges are Jordan arcs connecting different matrices (Fig. 1b). For clusters containing only two vertices, it is possible to show that the problem of computing planar NodeTrix representations coincides with the one of testing whether a special digraph, called the *canonical digraph*, associated to the network admits a 6-modal embedding. For higher values of modality, k -modal embeddings occur in the context of Intersection-Link representations of flat clustered networks. In an *intersection-link representation* [3, 5], vertices are represented as translates of the same polygon, intra-cluster edges are represented

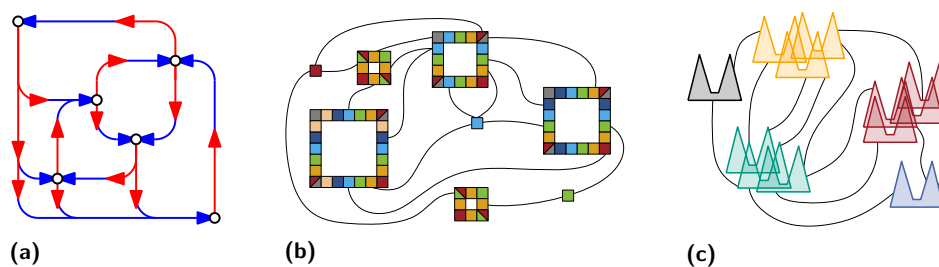


Figure 1 (a) A planar L-drawing, which determines a 4-modal embedding. (b) A planar NodeTrix representation. (c) A planar intersection-link representation using comb-shaped polygons.

via intersections between the polygons corresponding to their endpoints, and inter-cluster edges – similarly to NodeTrix representations – are Jordan arcs connecting the polygons corresponding to their endpoints. For any modality $k \geq 2$, it can be shown that testing the existence of a k -modal embedding of the canonical digraph of a flat clustered network with clusters of size two is equivalent to testing the existence of an intersection-link representation in which the curves representing inter-cluster edges do not intersect, when vertices are drawn as comb-shaped polygons (Fig. 1c).

Related Work. It is common knowledge that the existence of bimodal embeddings can be tested in linear time: Split each vertex v that has both incoming and outgoing edges into two vertices v_{in} and v_{out} , assign the incoming edges to v_{in} and the outgoing edges to v_{out} , connect v_{in} and v_{out} with an edge, and test the resulting (undirected) graph for planarity using any of the linear-time planarity-testing algorithms [11, 20]. Despite this, most of the planarity variants requiring bimodality are NP-complete; for instance, upward planarity [17], windrose planarity [6], partial-level planarity [12], clustered-level planarity and T -level planarity [4, 23], ordered-level planarity and bi-monotonicity [23]. In this scenario, a notable exception is represented by the classic level planarity problem, which can be solved in linear time [22], and its generalizations on the standing cylinder [7], rolling cylinder and the torus [2]. Although the existence of a bimodal embedding is easy to test, Binucci, Didimo, and Giordano [9] prove that the related problem of finding the maximum bimodal subgraph of an embedded planar digraph is an NP-hard problem. Moreover, Binucci, Didimo, and Patrignani [10] show that, given a *mixed planar graph*, i.e., a planar graph whose edge set is partitioned into a set of directed edges and a set of undirected edges, orienting the undirected edges in such a way that the whole graph admits a bimodal embedding is an NP-complete problem. On the other hand, the question regarding the computational complexity of constructing k -modal embedding for $k \geq 4$ has not been addressed, although the related problem of testing the existence of planar L-drawings has been recently proved NP-complete [13].

Our results. We study the complexity of the k -MODALITY problem, which asks for the existence of k -modal embeddings of planar digraphs – with an emphasis on $k = 4$. Our results are as follows:

- We demonstrate a complexity dichotomy for the 4-MODALITY problem with respect to the maximum degree Δ of the input digraph. Namely, we show NP-completeness when $\Delta \geq 7$ (see [21, Section 9]) and give a linear-time testing algorithm for $\Delta \leq 6$ (Theorem 19). Further, we extend the hardness result to any modality value larger than or equal to 4, by proving that the k -MODALITY problem is NP-complete for $k \geq 4$ when $\Delta \geq k + 3$.
- We provide an FPT-algorithm for k -MODALITY that runs in $f(k)O(n)$ time for the class of directed partial 2-trees (Theorem 16), which includes series-parallel and outerplanar digraphs.
- In Section 3, we relate k -modal embeddings with hybrid representations of flat clustered graphs, and exploit this connection to give new complexity results (Theorems 4 and 8) and algorithms (Theorems 3 and 7) for these types of representations. In particular, our NP-hardness results allow us to answer two open questions. Namely, we settle in the strongest possible way an open question, posed by Di Giacomo et al. [18, Open Problem (i)], about the complexity of computing planar NodeTrix representations of flat clustered graphs with clusters of size smaller than 5. Also, we address a research question by Angelini et al. [3, Open Problem (2)] about the representational power of intersection-link representations based on geometric objects that give rise to complex combinatorial structures, and solve it when the considered geometric objects are k -combs.
- Finally, in [21, Section 10], we show that not every outerplanar digraph admits a bimodal embedding, whereas any outerplanar (multi-)digraph admits a 4-modal embedding.

The algorithms presented in this paper employ the SPQ- and SPQR-tree data structures to succinctly represent the exponentially-many embeddings of series-parallel and biconnected planar digraphs, respectively, and can be easily modified to output an embedding of the input digraph in the same time bound. In particular, our positive result for $\Delta \leq 6$ is based on a set of simple reduction rules that exploit the structure of the rigid components of bounded-degree planar digraphs. These rules allow us to tackle the algorithmic core of the problem, by enabling a final reduction step to special instances of NAESAT, previously studied by Porschen et al. [26], which we prove to be always NAE-satisfiable [21, Section 7].²

2 Definitions

We assume familiarity with basic concepts concerning directed graphs, planar embeddings, connectivity and the BC-tree data structure; see the full version [21] for more details.

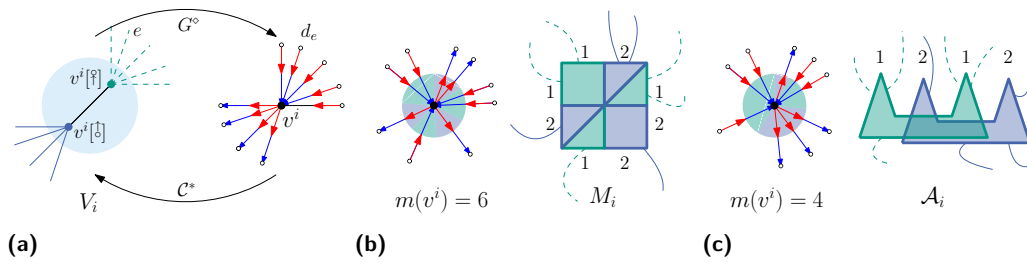
Directed graphs. A *directed graph* (for short *digraph*) $G = (V, E)$ is a pair, where V is the set of vertices and E is the set of (*directed*) *edges* of G , i.e., ordered pairs of vertices. We also denote the sets V and E by $V(G)$ and $E(G)$, respectively. The *underlying graph* of G is the undirected graph obtained from G by disregarding edge directions. Let v be a vertex, we denote by $E(v)$ the set of edges of G incident to v and by $\deg(v) = |E(v)|$ the *degree* of v . For an edge $e = uv$ directed from u to v and an end-point $x \in \{u, v\}$ of e , we define the *orientation* $\sigma(e, x)$ of e at x as $\sigma(e, x) = \uparrow$, if $x = u$, and $\sigma(e, x) = \downarrow$, if $x = v$, and we say that uv is *outgoing from* u and *incoming at* v .

Modality. Let G be a planar digraph and let \mathcal{E} be an embedding of G . A pair of edges e_1, e_2 that appear consecutively in the circular order around a vertex v of G is *alternating* if they do not have the same orientation at v , i.e., they are not both incoming at or both outgoing from v . Also, we say that vertex v is *k -modal*, or that v has *modality k* , or that the *modality of v is k* in \mathcal{E} , if there exist exactly k alternating pairs of edges incident to v in \mathcal{E} . Clearly, the value k needs to be a non-negative even integer. An embedding of a digraph G is *k -modal*, if each vertex is at most k -modal; see Fig. 3(left).

We now define an auxiliary problem, called k -MAXMODALITY (where k is a positive even integer), which will be useful to prove our algorithmic results. We denote the set of non-negative integers by \mathbb{Z}^* and the set of non-negative even integers smaller than or equal to k as $\mathbb{E}_k^+ = \{b : b = 2a, b \leq k, a \in \mathbb{Z}^*\}$. Given a graph G , we call *maximum-modality function* an integer-valued function $m : V(G) \rightarrow \mathbb{E}_k^+$. We say that an embedding \mathcal{E} of G *satisfies m at a vertex v* if the modality of v in \mathcal{E} is at most $m(v)$.

Problem: k -MAXMODALITY
Input: A pair $\langle G, m \rangle$, where G is a digraph and m is a maximum-modality function.
Question: Is there an embedding \mathcal{E} of G that <i>satisfies</i> m at every vertex?

² In “Stefan Porschen, Bert Randerath, Ewald Speckenmeyer: Linear Time Algorithms for Some Not-All-Equal Satisfiability Problems. SAT 2003: 172-187” [26], the authors state in the abstract “First we show that a NAESAT model (if existing) can be computed in linear time for formulas in which each variable occurs at most twice.” We give a strengthening of this result by showing that the only negative formulas with the above properties are those whose variable-clause graph contains components isomorphic to a simple cycle and provide a recursive linear-time algorithm for computing a NAE-truth assignment for formulas in which each variable occurs at most twice, when one exists, which is also considerably simpler than the one presented in [26].



■ **Figure 2** (a) Illustrations for the duality between the canonical digraph and the canonical c-graph. Correspondence (b) between 6-modal embeddings and planar NodeTrix representations, and (c) between 4-modal embeddings and clique-planar representations using 2-combs as geometric objects.

3 Implications on Hybrid Representations

A *flat clustered graph* (for short, *c-graph*) is a pair $\mathcal{C} = (G = (V, E), \mathcal{P} = (V_1, V_2, \dots, V_c))$, where G is a graph and \mathcal{P} is a partition of V into sets V_i , for $i = 1, \dots, c$, called *clusters*. An edge $(u, v) \in E$ with $u \in V_i$ and $v \in V_j$ is an *intra-cluster edge*, if $i = j$, and is an *inter-cluster edge*, if $i \neq j$. The problem of visualizing such graphs so to effectively convey both the relation information encoded in the set E of edges of G and the hierarchical information given by the partition \mathcal{P} of the clusters has attracted considerable research attention. In a *hybrid representation* of a graph different conventions are used to represent the dense and the sparse portions of the graph [3, 5, 14, 18, 19, 24, 27]. We present important implications of our results on some well-known models for hybrid-representations of c-graphs.

Let \mathcal{C} be a c-graph whose every cluster forms a clique of size at most 2, that is, each cluster contains at most two vertices connected by an intra-cluster edge. Starting from \mathcal{C} we define an auxiliary digraph G° , called the *canonical digraph* for \mathcal{C} , as follows. Without loss of generality, assume that, for $i = 1, \dots, c$, each cluster V_i contains two vertices denoted as $v^i[\uparrow]$ and $v^i[\downarrow]$. The vertex set of G° contains a vertex v^i , for $i = 1, 2, \dots, c$, and a dummy vertex d_e , for each inter-cluster edge $e \in E$. The edge set of G° contains two directed edges, for each inter-cluster edge $e = (v_x^i, v_y^j) \in E$, with $x, y \in \{\uparrow, \downarrow\}$ and $i \neq j$; namely, $E(G^\circ)$ contains (i) either the directed edges $v_x^i d_e$, if $x = \downarrow$, or the directed edge $d_e v_x^i$, if $x = \uparrow$, and (ii) either the directed edges $v_y^j d_e$, if $y = \downarrow$, or the directed edge $d_e v_y^j$, if $y = \uparrow$.

Let now $D = (V, E)$ be a digraph. We construct a c-graph $\mathcal{C}^* = (G^* = (V^*, E^*), \mathcal{P}^*)$ from D whose every cluster forms a clique of size at most 2, called the *canonical c-graph* for D , as follows. For each vertex $v^i \in V$, G^* contains two vertices $v^i[\uparrow]$ and $v^i[\downarrow]$, which form the cluster $V_i = \{v^i[\uparrow], v^i[\downarrow]\}$ in \mathcal{P}^* . For each (directed) edge $v^i v^j$ of D , G^* contains an (undirected) edge $(v^i[\downarrow], v^j[\uparrow])$; that is, each directed edge in E that is incoming (outgoing) at a vertex v^i and outgoing (incoming) at a vertex v^j corresponds to an inter-cluster edge in E^* incident to $v^i[\uparrow]$ (to $v^i[\downarrow]$) and to $v^j[\downarrow]$ (to $v^j[\uparrow]$). Finally, for each vertex $v^i \in V$, G^* contains an intra-cluster edge $(v^i[\uparrow], v^i[\downarrow])$. The canonical digraph and the canonical c-graph form dual concepts, as illustrated in Fig. 2a; the canonical c-graph of G° is the original c-graph \mathcal{C} (neglecting clusters originated by dummy vertices) and the canonical digraph of \mathcal{C}^* is the original digraph D (suppressing dummy vertices).

NodeTrix Planarity. A *NodeTrix representation* of a c-graph $\mathcal{C} = (G, \mathcal{P})$ is a drawing of \mathcal{C} such that: (i) each cluster $V_i \in \mathcal{P}$ is represented as a symmetric adjacency matrix M_i (with $|V_i|$ rows and columns), drawn in the plane so that its boundary is a square Q_i with sides parallel to the coordinate axes; (ii) no two matrices intersect, that is, $Q_i \cap Q_j = \emptyset$, for

all $1 \leq i < j \leq c$; (iii) each intra-cluster edge is represented by the adjacency matrix M_i ; and (iv) each inter-cluster edge (u, v) with $u \in V_i$ and $v \in V_j$ is represented as a simple Jordan arc connecting a point on the boundary of Q_i with a point on the boundary of Q_j , where the point on Q_i (on Q_j) belongs to the column or to the row of M_i (resp. of M_j) associated with u (resp. with v). A NodeTrix representation is *planar* if no inter-cluster edge intersects a matrix or another inter-cluster edge, except possibly at a common end-point; see Figs. 1b and 2b. The NODETRIX PLANARITY problem asks whether a c-graph admits a planar NodeTrix representation. NODETRIX PLANARITY has been proved NP-complete for c-graphs whose clusters have size larger than or equal to 5 [18].

We are ready to establish our main technical lemmas.

- **Lemma 1.** *C-graph \mathcal{C} is NodeTrix planar if and only if G^\diamond admits a 6-modal embedding.*
- **Lemma 2.** *Digraph D admits a 6-modal embedding if and only if \mathcal{C}^* is NodeTrix planar.*

Proof sketch for Lemmas 1 and 2. Let M_i be the matrix representing cluster $V_i = \{v^i[\uparrow], v^i[\downarrow]\}$. We have that, independently of which of the two possible permutations for the rows and columns of M_i is selected, the boundary of Q_i is partitioned into three maximal portions associated with $v^i[\uparrow]$ and three maximal portions associated with $v^i[\downarrow]$; that is, they form the pattern [1, 2, 1, 2, 1, 2], see Fig. 2b. Therefore, any planar NodeTrix representation of \mathcal{C} (of \mathcal{C}^*) can be turned into a 6-modal embedding of G^\diamond (of D) via a local redrawing procedure which operates in the interior of Q_i ; also, any 6-modal embedding of G^\diamond (of D) can be turned into a planar NodeTrix representation of \mathcal{C} (of \mathcal{C}^*) via a local redrawing procedure which operates in a small disk centered at v_i that contains only v_i and intersects only edges incident to v_i .

Since G^\diamond can be constructed in linear time from \mathcal{C} , Lemma 1 and the algorithm of Theorem 16 for solving k -MODALITY of directed partial 2-trees give us the following.

- **Theorem 3.** *NODETRIX PLANARITY can be solved in linear time for flat clustered graphs whose clusters have size at most 2 and whose canonical digraph is a directed partial 2-tree.*

Note that (i) \mathcal{C}^* can be constructed in polynomial time from D , (ii) \mathcal{C}^* only contains clusters of size 2 (although clusters corresponding to vertices of D incident to incoming or outgoing edges only could be simplified into clusters of size 1), and (iii) each cluster $V_i \in \mathcal{P}^*$, with $v^i \in V(D)$, is incident to α inter clusters edges, where α is the degree of v^i in D . These properties and the fact that in [21, Section 9] we prove the k -MODALITY problem to be NP-complete for digraphs of maximum degree $\Delta \geq k + 3$ give us the following.

- **Theorem 4.** *NODETRIX PLANARITY is NP-complete for flat clustered graphs whose clusters have size at most 2, even if each cluster is incident to at most 9 inter-cluster edges.*

We remark that the above NP-completeness result is best possible in terms of the size of clusters, as clusters of size 1 do not offer any advantage to avoid intersections between inter-cluster edges. Also, it solves [18, Open Problem (i)], which asks for the complexity of NODETRIX PLANARITY for c-graphs whose clusters have size between 2 and 5.

Clique Planarity. Hybrid representations have also been recently studied in the setting in which clusters are represented via intersections of geometric objects. In particular, Angelini et al. [3] introduced the following type of representations. Suppose that a c-graph (G, \mathcal{P}) is given, where \mathcal{P} is a set of cliques that partition the vertex set of G . In an *intersection-link representation*, the vertices of G are represented by geometric objects that are translates

of the same rectangle. Consider an edge (u, v) and let $R(u)$ and $R(v)$ be the rectangles representing u and v , respectively. If (u, v) is an intra-cluster edge (called *intersection-edge* in [3]), we represent it by drawing $R(u)$ and $R(v)$ so that they intersect, otherwise if (u, v) is an inter-cluster edge (called *link-edge* in [3]), we represent it by a Jordan arc connecting $R(u)$ and $R(v)$. A *clique-planar* representation is an intersection-link representation in which no inter-cluster edge intersects the interior of any rectangle or another inter-cluster edge, except possibly at a common end-point. The CLIQUE PLANARITY problem asks whether a c-graph (G, \mathcal{P}) admits a clique-planar representation.

Angelini et al. proved the CLIQUE PLANARITY problem to be NP-complete, when \mathcal{P} contains a cluster V^* with $|V^*| \in O(|G|)$, and asked, in [3, Open Problem (2)], about the implications of using different geometric objects for representing vertices, rather than translates of the same rectangle. We address this question by considering k -combs as geometric objects, where a k -comb is the simple polygon with k spikes illustrated in Fig. 2c. We have the following.

► **Lemma 5.** *C-graph \mathcal{C} is a positive instance of CLIQUE PLANARITY using k -combs as geometric objects if and only if G^\diamond admits a $2k$ -modal embedding.*

► **Lemma 6.** *Digraph D admits an 4-modal embedding if and only if \mathcal{C}^* is a positive instance of CLIQUE PLANARITY using 2-combs as geometric objects.*

Proof sketch for Lemmas 5 and 6. Let A_i be an arrangements of 2-combs representing cluster $V_i = \{v^i[\uparrow], v^i[\downarrow]\}$. We have that, the boundary of A_i is partitioned into at most two maximal portions associated with $v^i[\uparrow]$ and at most two maximal portions associated with $v^i[\downarrow]$; that is, they form the pattern $[1, 2, 1, 2]$, see Fig. 2c. Therefore, as for Lemmas 1 and 2, we can exploit a local redrawing procedure to transform a clique-planar representation of \mathcal{C} (of \mathcal{C}^*) into a 4-modal embedding of G^\diamond (of D), and vice versa.

Combining Lemma 5 and the algorithm of Theorem 16 gives us the following positive result.

► **Theorem 7.** *CLIQUE PLANARITY using r -combs, with $r \geq 1$, as geometric objects can be solved in linear time for flat clustered graphs whose clusters have size at most 2 and whose canonical digraph is a directed partial 2-tree.*

Finally, Lemma 6 and the discussion preceding Theorem 4 imply the following.

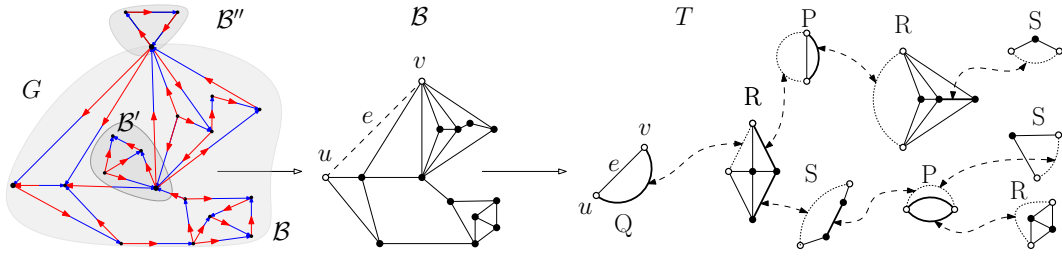
► **Theorem 8.** *CLIQUE PLANARITY using 2-combs as geometric objects is NP-complete, even for flat clustered graphs with clusters of size at most 2 each incident to at most 7 inter-cluster edges.*

4 Polynomial-time Algorithms

In this section, we present an algorithmic framework to devise efficient algorithms for the k -MODALITY problem for notable families of instances. First, in Section 4.1, we show how to efficiently reduce the k -MODALITY problem in simply-connected digraphs to the k -MAXMODALITY problem in biconnected digraphs. Then, in Section 4.2, we introduce preliminaries and definitions concerning SPQR-trees and k -modal embeddings of biconnected digraphs.

4.1 Simply-Connected Graphs

We first observe that the k -MAXMODALITY problem is a generalization of the k -MODALITY problem. In fact, a directed graph $G = (V, E)$ admits a k -modal embedding if and only if the pair $\langle G, m \rangle$, with $m(v) = k, \forall v \in V(G)$, is a positive instance of the k -MAXMODALITY problem.



■ **Figure 3** (left) A 4-modal embedding of a simply-connected planar digraph G . (right) The SPQR tree \mathcal{T} of the block β of G rooted at the edge $e = uv$. The extended skeletons of all non-leaf nodes of \mathcal{T} are shown; virtual edges corresponding to S-, P-, and R-nodes are thick.

► **Observation 1.** k -MODALITY reduces in linear time to k -MAXMODALITY.

Let $\langle G, m : V(G) \rightarrow \mathbb{E}_4^+ \rangle$ be an instance of 4-MAXMODALITY; also, let β be a leaf-block of the BC-tree \mathcal{T} of G and let v be the parent cut-vertex of β in \mathcal{T} . We denote by G_β^- the subgraph of G induced by v and the vertices of G not in β , i.e., $G_\beta^- = G - (\beta - \{v\})$. Also, let $B(\mathcal{T})$ be the set of blocks in \mathcal{T} . We show that k -MAXMODALITY (and k -MODALITY, by Observation 1) in simply-connected digraphs is Turing reducible to k -MAXMODALITY in biconnected digraphs.

► **Theorem 9.** *Given a subroutine TESTBICONNECTED that tests k -MAXMODALITY for biconnected instances, there exists a procedure TESTSIMPLYCONNECTED that tests k -MAXMODALITY for simply-connected digraphs. Further, given an instance $\langle G, m \rangle$ of k -MAXMODALITY, the runtime of TESTSIMPLYCONNECTED($\langle G, m \rangle$) is*

$$O(|G| + \log k \sum_{\beta \in B(\mathcal{T})} r(\beta)),$$

where $r(\beta)$ is the runtime of TESTBICONNECTED($\langle \beta, m \rangle$) and \mathcal{T} is the BC-tree of G .

Sketch. The algorithm selects a leaf-block β of \mathcal{T} , with parent cut-vertex v , and finds an embedding of β with the minimum modality at v satisfying m at all vertices, by performing a binary search using the TESTBICONNECTED procedure. We then remove β , replace G with G_β^- , and update the value of $m(v)$, so to account for the alternations at v introduced by β . The procedure terminates when all the blocks have been processed. Therefore, since the total number of calls to subroutine TESTBICONNECTED is bounded by the number of blocks of G , which is $O(|\mathcal{T}|) = O(|G|)$ multiplied by $\log k$, the overall running time is $O(|G| + \log k \sum_{\beta \in B(\mathcal{T})} r(\beta))$. ◀

4.2 Biconnected Graphs

To handle the decomposition of a biconnected digraph into its triconnected components, we use SPQR-trees, a data structure introduced by Di Battista and Tamassia [16].

SPQR-trees. Let G be a biconnected digraph. We consider SPQR-trees that are rooted at an edge e of G , called the *reference edge*. The rooted SPQR-tree \mathcal{T} of G with respect to e describes a recursive decomposition of G induced by its split pairs. The nodes of \mathcal{T} are of four types: S, P, Q, and R. Each node μ of \mathcal{T} has an associated undirected multigraph $\text{skel}(\mu)$, called the *skeleton* of μ , with two special nodes u_μ and v_μ (the *poles* of μ), and an associated subgraph $\text{pert}(\mu)$ of G , called *pertinent* of μ . The skeleton graph equipped with the edge $u_\mu v_\mu$, called the *parent edge*, is the *extended skeleton* of μ . Refer to Fig. 3(right).

Each edge of $\text{skel}(\mu)$, called *virtual edge*, is associated with a child of μ in \mathcal{T} . The skeleton of μ describes how the pertinent graphs of the children of μ have to be “merged” via their poles to obtain $\text{pert}(\mu)$. The extended skeleton of an S-, P-, R-, and Q-node is a cycle, parallel, triconnected graph, and a 2-gon, respectively. It follows that skeleton and pertinent graphs are always biconnected once the parent edge is added. A *series-parallel digraph* is a biconnected planar digraph whose SPQR-tree only contains S-, P-, and Q-nodes. A *partial 2-tree* is a digraph whose every block is a series-parallel digraph.

A digraph G is planar if and only if the skeleton of each R-node in the SPQR-tree of G is planar. By selecting *regular embeddings* for the skeletons of the nodes of \mathcal{T} , that is, embeddings in which the parent edge is incident to the outer face, we can construct any embedding of G with the edge e on the outer face, where the choices for the embeddings of the skeletons are all and only the (i) flips of the R-nodes and the (ii) permutations for the virtual edges of the P-nodes.

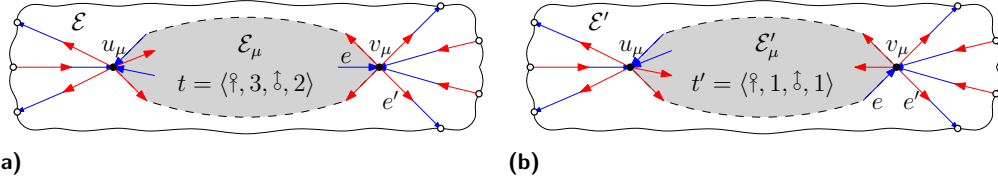
Consider a pair $\langle G, m \rangle$ such that G is biconnected and let \mathcal{E} be a planar embedding of G . Also, let \mathcal{T} be the SPQR-tree of G rooted at an edge e of G incident to the outer face of \mathcal{E} . We will assume that the virtual edges of the skeletons of the nodes in \mathcal{T} are oriented so that the extended skeleton of each node μ is a DAG with a single source u_μ and a single sink v_μ . Let μ be a node of \mathcal{T} and let \mathcal{E}_μ be the planar (regular) embedding of $\text{skel}(\mu)$ induced by \mathcal{E} . For an oriented edge $d = uv$ of $\text{skel}(\mu)$, the *left* and *right face* of d in \mathcal{E}_μ is the face of \mathcal{E}_μ seen to the left and to the right of d , respectively, when traversing this edges from u to v . We define the *outer left (right) face* of \mathcal{E}_μ as the left (right) face of the edge $u_\mu v_\mu$ in \mathcal{E}_μ .

Embedding tuples. An *embedding tuple* (for short, *tuple*) is a 4-tuple $\langle \sigma_1, a, \sigma_2, b \rangle$, where $\sigma_1, \sigma_2 \in \{\uparrow, \uparrow\}$ are orientations and $a, b \in \mathbb{N}$ are non-negative integers. Consider two tuples $t = \langle \sigma_1, a, \sigma_2, b \rangle$ and $t' = \langle \sigma'_1, a', \sigma'_2, b' \rangle$. We say that t *dominates* t' , denoted as $t \preceq t'$, if $\sigma_1 = \sigma'_1$, $\sigma_2 = \sigma'_2$, $a \leq a'$, and $b \leq b'$. Also, we say that t and t' are *incompatible*, if none of them dominates the other. Since the relationship \preceq is reflexive, antisymmetric, and transitive, it defines a poset (T, \preceq) , where T is the set of embedding tuples. A subset $S \subseteq T$ is *succinct* or an *antichain*, if the tuples in S are pair-wise incompatible. Consider two subsets $S, S' \subseteq T$ of tuples. We say that S *dominates* S' , denoted as $S \preceq S'$, if for any tuples $t' \in S'$ there exists at least one tuple $t \in S$ such that $t \preceq t'$. Also, S *reduces* S' if $S \preceq S'$ and $S \subseteq S'$. Finally, S is a *gist* of S' , if S is succinct and reduces S' .

Let e_u and e_v be the edges of $\text{pert}(\mu)$ incident to the outer left face of \mathcal{E}_μ and to u_μ and v_μ , respectively, possibly $e_u = e_v$. Also, let a and b be non-negative integers. We say that the embedding \mathcal{E}_μ *realizes* tuple $\langle \sigma_1, a, \sigma_2, b \rangle$, if $\sigma_1 = \sigma(e_u, u_\mu)$, $\sigma_2 = \sigma(e_v, v_\mu)$, and a and b are the number of inner faces of \mathcal{E}_μ whose (two) edges incident to u_μ and to v_μ , respectively, form an alternating pair. A tuple $t = \langle \sigma_1, a, \sigma_2, b \rangle$ is *realizable by* μ , if there exists an embedding of $\text{pert}(\mu)$ that realizes t , and *admissible*, if $a \leq m(u)$ and $b \leq m(v)$. A tuple is *good for* μ if it is both admissible and realizable by μ . We denote by $S(\mu)$ the gist of the set of good tuples for a node μ . Let e_μ be the virtual edge representing μ in the skeleton of the parent of μ in \mathcal{T} , with a small overload of notation, we also denote $S(\mu)$ by $S(e_\mu)$. For a tuple $t = \langle \sigma_1, a, \sigma_2, b \rangle \in S(e_\mu)$, where $e = u_\mu v_\mu$, the pair (σ_1, a) is the *embedding pair* of t at u_μ ; likewise, the pair (σ_2, b) is the *embedding pair* of t at v_μ . We have the following substitution lemma.

► **Lemma 10.** *Let \mathcal{E} be a planar embedding of G satisfying m . Let μ be a node of \mathcal{T} and let \mathcal{E}_μ be the embedding of $\text{pert}(\mu)$ induced by \mathcal{E} . Also, let $\mathcal{E}'_\mu \neq \mathcal{E}_\mu$ be an embedding of $\text{pert}(\mu)$ satisfying m . Then, G admits an embedding \mathcal{E}' satisfying m in which the embedding of $\text{pert}(\mu)$ is \mathcal{E}'_μ , if $t' \preceq t$, where t and t' are the embedding tuples realized by \mathcal{E}_μ and by \mathcal{E}'_μ , respectively.*

19:10 Computing k -Modal Embeddings of Planar Digraphs



■ **Figure 4** Illustration for the proof of Lemma 10. The parity of t and t' is the same at u_μ and different at v_μ ; in particular, even if a new alternation is introduced between the pair (e, e') at v_μ , the different parity guarantees that the modality at v_μ does not increase from \mathcal{E} to \mathcal{E}' .

Sketch. We show how to construct a drawing Γ'_G of G satisfying m in which the embedding of $\text{pert}(\mu)$ is \mathcal{E}'_μ ; see Fig. 4. Let Γ_G be a drawing of G whose embedding is \mathcal{E} . Remove from Γ_G the drawing of all the vertices of $\text{pert}(\mu)$ different from u_μ and v_μ and the drawing of all the edges of $\text{pert}(\mu)$. Denote by f the face of the resulting embedded graph G^- that used to contain the removed vertices and edges. We obtain Γ'_G by inserting a drawing of $\text{pert}(\mu)$ whose embedding is \mathcal{E}'_μ in the interior of f so that vertices u_μ and v_μ are identified with their copies in G^- . We can prove that the embedding \mathcal{E}' of Γ'_G satisfies m by exploiting the interplay between the parity and the number of alternations at u_μ (at v_μ) in t' and t when $t' \preceq t$. ◀

Let \mathcal{T} be the SPQR-tree \mathcal{T} of G rooted at a reference edge e . In the remainder of the section, we show how to compute the gist $S(\mu)$ of the set of good tuples for μ , for each non-root node μ of \mathcal{T} . In the subsequent procedures to compute $S(\mu)$ for S-, P-, and R-nodes, we are not going to explicitly avoid set $S(\mu)$ to contain dominated tuples. In fact, this can always be done at the cost of an additive $O(k^2)$ factor in the running time, by maintaining an hash table that stores the tuples that have been constructed (possibly multiple times) by the procedures and by computing the gist of the constructed set as a final step.

► **Property 1.** For each node $\mu \in \mathcal{T}$, it holds that $|S(\mu)| \in O(k)$.

Proof. By the definition of gist, any embedding pair (σ, a) has at most two tuples $t', t'' \in S(\mu)$ such that (σ, a) is the embedding pair of t' and t'' at u_μ ; also, the embedding pairs (σ', a') of t' and (σ'', a'') of t'' at v_μ are such that $\sigma' \neq \sigma''$. Since there exist at most $2k$ realizable embedding pairs (σ, a) at u_μ (as $\sigma \in \{\ddagger, \dagger\}$, $a \in \{0, 1, \dots, k\}$), and the existence of tuple whose embedding pair at u_μ is $(\sigma, 0)$ implies that all tuples have such an embedding pair at u_μ , we have $|S(\mu)| \leq 4k$. ◀

If μ is a leaf Q-node of \mathcal{T} , then $S(\mu) = \{\langle \sigma(u_\mu v_\mu, u_\mu), 0, \sigma(u_\mu v_\mu, v_\mu), 0 \rangle\}$. If μ is an internal node of \mathcal{T} , we visit \mathcal{T} bottom-up and compute the set $S(\mu)$ for μ assuming to have already computed the sets $S(\mu_1), \dots, S(\mu_k)$ for the children μ_1, \dots, μ_k of μ (where μ_i is the child of μ corresponding to the edge e_i in $\text{skel}(\mu)$). Let ρ be the unique child of the root of \mathcal{T} . Once the set $S(\rho)$ has been determined, we can efficiently decide whether G admits an embedding satisfying m in which the reference edge e is incident to the outer face by means of the following lemma.

► **Lemma 11.** Given $S(\rho)$, we can test whether G has an embedding that satisfies m in $O(k^2)$ time.

5 Partial 2-trees

In the following, we describe how to compute $S(\mu)$, if μ is an S-node (Lemma 12) and a P-node (Lemma 13) in $O(f(k)|\text{skel}(\mu)|)$ time, where f is a computable function.

► **Lemma 12.** *Set $S(\mu)$ can be constructed in $O(k^2|\text{skel}(\mu)|)$ time for an S-node μ .*

Sketch. Let μ be an S-node with skeleton $\text{skel}(\mu) = (e_1, e_2, \dots, e_h)$. We define τ_j as the S-node obtained by the series composition of $\mu_1, \mu_2, \dots, \mu_j$, with $j \leq h$. Initially we set $S(\tau_1) = S(e_1)$. Then, we construct $S(\tau_j)$, for $j = 2, \dots, h$, by verifying the compatibility of the embedding pairs of the good tuples of the virtual edges of $\text{skel}(\tau_j)$ at the internal vertices of $\text{skel}(\tau_j)$. As $S(\tau_j) = S(\tau_{j-1}) \cup e_j$, we can compute $S(\tau_j)$ by considering all the tuples obtained by combining every tuple $t' \in S(\tau_{j-1})$ with every tuple $t'' \in S(e_j)$. Since both these sets contain $O(k)$ tuples, by Property 1, and since the tuple resulting from the combination of t' and t'' can be determined in $O(1)$ time, we have that $S(\tau_j)$ can be computed in $O(k^2)$ time. Therefore, the overall running time for computing $S(\mu) = S(\tau_h)$ is $O(k^2|\text{skel}(\mu)|)$. ◀

► **Lemma 13.** *Set $S(\mu)$ can be constructed in $O((2k+4)!k^3 + |\text{skel}(\mu)|)$ time for a P-node μ .*

Sketch. Let μ be a P-node with poles u_μ and v_μ , whose skeleton $\text{skel}(\mu)$ consists of h parallel virtual edges e_1, e_2, \dots, e_h . It can be shown that the computation of $S(\mu)$ reduces in $O(|\text{skel}(\mu)|)$ time to the computation of $S(\tau)$, where τ is a P-node whose skeleton consists of at most $2k$ virtual edges of $\text{skel}(\mu)$ that contribute with at least one alternating pair of edges at u_μ or v_μ , plus up to 4 virtual edges of $\text{skel}(\mu)$ that contribute with no alternating pair at u_μ or at v_μ . For any permutation π of the virtual edges of $\text{pert}(\tau)$, let τ_i^π be the P-node obtained by restricting τ to the first i virtual edges in π . We fix the embedding of $\text{skel}(\tau_i^\pi)$ in such a way that the virtual edges of $\text{skel}(\tau_i^\pi)$ are ordered according to π . Then, in a fashion similar to the S-node case, we can compute $S(\tau_i^\pi)$ for the given embedding of $\text{skel}(\tau_i^\pi)$ by combining $S(\tau_{i-1}^\pi)$ and $S(e_i)$ in $O(k^2)$ time (recall that both these sets have size $O(k)$, by Property 1). Clearly, for any fixed π , we can compute $S(\tau_\pi^h)$ in $O(k^3)$ time. Thus, by performing the above computation for all the $(2k+4)!$ possible permutations for the virtual edges of $\text{pert}(\tau)$, we can construct $S(\tau)$ in $O((2k+4)!k^3 + |\text{skel}(\mu)|)$ time. ◀

Altogether, Lemmas 12 and 13 yield the following main result.

► **Lemma 14.** *k -MAXMODALITY can be solved in $O((2k+4)!k^3n)$ for series-parallel digraphs.*

Observation 1, Lemma 14, and Theorem 9 immediately imply the following.

► **Corollary 15.** *k -MODALITY can be solved in $O(((2k+4)!k^3 \log k)n)$ for directed partial 2-trees.*

Due to the special algorithmic framework we are employing, we can however turn the multiplicative $O(\log k)$ factor in the running time into an additive $O(k)$ factor by modifying Theorem 9 as follows. When considering a cut-vertex v , we will execute “only once” the function TESTBICONNECTED by rooting the SPQ-tree at a Q-node η corresponding to an edge incident to v . This will allow us to compute the minimum modality for cut-vertex v in an embedding that satisfies m at every vertex, by simply scanning the set $S(\eta)$, which takes $O(k)$ time by Property 1, rather than by exploiting a logarithmic number of calls to TESTBICONNECTED.

► **Theorem 16.** *k -MODALITY can be solved in $O((2k+4)!k^3n)$ for directed partial 2-trees.*

6 A Linear-time Algorithm for 4-MaxModality when $\Delta \leq 6$

In this section, we show that in the special case when $k = 4$ and G has maximum degree $\Delta \leq 6$, it is possible to compute the set $S(\mu)$ when μ is an R-node in linear time in the size of $\text{skel}(\mu)$.

Our strategy to compute $S(\mu)$ is as follows. We select a single tuple from the admissible set of each virtual edge incident to u_μ and v_μ , in every possible way. Each selection determines a “*candidate tuple*” t for $S(\mu)$. First, we check if t is admissible at both u and v . Second, we restrict the tuples of the edges incident to the poles to only the tuples that form t and check if there is a way of satisfying m at the (inner) vertices of $\text{skel}(\mu)$. If both the poles and the inner vertices are satisfiable, then we add t to $S(\mu)$. Since the degrees of the poles are bounded, there is at most a constant number of candidate tuples which must be checked. The complexity lies in this check.

We now formally describe how to compute $S(\mu)$. First, for each virtual edge e_i of $\text{skel}(\mu)$ incident to the poles of μ , we select a tuple t_i from $S(\mu_i)$. Let $T_u = [t_{u,1}, t_{u,2}, \dots, t_{u,\ell}]$ and $T_v = [t_{v,1}, t_{v,2}, \dots, t_{v,h}]$ be the list of tuples selected for the virtual edges incident to u_μ and to v_μ , respectively. Each pair of lists T_u and T_v yields a *candidate tuple* $t = \langle \sigma_1, a, \sigma_2, b \rangle$ for μ . However, the tuples selected to construct T_u and T_v allow for an admissible embedding of $\text{pert}(\mu)$ realizing tuple t *if and only if*: **(Condition 1)** tuple t satisfies m at u_μ and at v_μ , and **(Condition 2)** it is possible to select tuples for each of the remaining virtual edges of $\text{skel}(\mu)$ that satisfy m at every internal vertex of $\text{skel}(\mu)$. Let $\mathcal{P}(\mu)$ be the set of candidate tuples for μ constructed as described above. We can easily filter out the candidate tuples that do not satisfy Condition 1. In the remainder of the section, for each pair of lists T_u and T_v yielding a tuple $t \in \mathcal{P}(\mu)$, we will show how to test Condition 2 for μ in linear time. This and the fact that $|\mathcal{P}(\mu)| \in O(1)$ imply the following.

► **Lemma 17.** *Set $S(\mu)$ can be constructed in $O(|\text{skel}(\mu)|)$ time for an R-node μ , if $\Delta \leq 6$.*

Altogether, Lemmas 12, 13 and 17 yield the following main result.

► **Lemma 18.** *4-MAXMODALITY can be solved in linear time for biconnected digraphs with $\Delta \leq 6$.*

Observation 1, Lemma 18, and Theorem 9 immediately imply the following.

► **Theorem 19.** *4-MODALITY can be solved linear time for digraphs with $\Delta \leq 6$.*

To prove Lemma 17, we show how to solve the following auxiliary problem for special instances.

Problem: 4-MAXSKELMODALITY
<p>Input: A triple $\langle G = (V, E), \mathcal{S} = \{S(e_1), \dots, S(e_{ E })\}, m \rangle$ where G is an embedded directed graph, each $S(e_i)$ is a set containing embedding tuples for the virtual edge $e_i \in E$, and $m : V \rightarrow \mathbb{E}_4^+$ is the <i>maximum-modality function</i>.</p> <p>Question: Can we select a tuple from each set $S(e_i)$ in such a way that the modality at each vertex $v \in V$ is at most $m(v)$?</p>

For each pair of lists T_u and T_v yielding a candidate tuple in $\mathcal{P}(\mu)$, we will construct an instance $I_\mu(T_u, T_v) = (G, \mathcal{S}, m)$ of 4-MAXSKELMODALITY as follows. First, we set $G = \text{skel}(\mu)$ and we fix the embedding of G to be equal to the unique regular embedding of $\text{skel}(\mu)$. Second, for each virtual edge $e_{u,i}$ incident to u_μ , with $i = 1, \dots, \ell$, we set $S(e_{u,i}) = \{t_{u,i}\}$; for each virtual edge $e_{v,j}$ incident to v_μ , with $j = 1, \dots, h$, we set $S(e_{v,j}) = \{t_{v,j}\}$; and, for each of the remaining virtual edges e_d of $\text{skel}(\mu)$, we set $S(e_d) = S(\mu_d)$. Finally, the maximum-modality function of I_μ coincides with m .

Clearly, $I_\mu(T_u, T_v)$ is a positive instance of 4-MAXSKELMODALITY if and only if, given the constraints imposed by the tuples in T_u and in T_v , there exists a selection of tuples for the edges of G not incident to u_μ or v_μ that satisfies m at all the internal vertices of G , i.e., Condition 2 holds.

Let v be a vertex of G and let e be an edge in $E(v)$, we denote by $A_v(e)$ the maximum number of alternations at v over all the tuples in $S(e)$.

► **Definition 20** (Good instances). *An instance of 4-MAXSKELMODALITY is good if, for any vertex v in G , it holds $\sum_{e \in E(v)} (A_v(e) + 1) \leq 6$.*

Note that, for each edge e in $\text{ske}(\mu)$ incident to a vertex v , $\text{pert}(e)$ contributes at least $A_v(e) + 1$ edges to $d_{\text{pert}(e)}(v)$. Thus, we have $\sum_{e \in E(v)} (A_v(e) + 1) \leq \sum_{e \in E(v)} d_{\text{pert}(e)}(v) \leq 6$. Therefore, instance $I_\mu(T_u, T_v)$ is good. Although 4-MAXSKELMODALITY turns out to be NP-complete in general [21, Section 9], we are now going to show the following main positive result.

► **Theorem 21.** *4-MAXSKELMODALITY is linear-time solvable for good instances.*

The outline of the linear-time algorithm to decide whether a good instance $I = \langle G = (V, E), \mathcal{S} = \{S(e_1), \dots, S(e_{|E|})\}, m \rangle$ of 4-MAXSKELMODALITY is a positive instance is as follows.

- We process I by means of a set of *reduction rules* applied locally at the vertices of G and their incident edges. Each of these rules, if applicable, either detects that the instance I is a negative instance or transforms it into an equivalent smaller instance $I' = \langle G', \mathcal{S}', m' \rangle$. Each rule can be applied when specific conditions are satisfied at the considered vertex. A rule may additionally set a vertex as *marked*. Any marked vertex v has the main property that **any selection** of tuples from the admissible sets of the edges incident to v satisfies m' at v .
- Let I^* be the instance of 4-MAXSKELMODALITY obtained when no reduction rule may be further applied. We prove that instance I^* has a special structure that allows us to reduce the problem of testing whether I^* is a positive instance of 4-MAXSKELMODALITY to that of verifying the NAE-satisfiability of a constrained instance of NAESAT, in fact, of PLANAR NAESAT. Since PLANAR NAESAT is in P [25], this immediately implies that 4-MAXSKELMODALITY is also in P. However, in [21, Section 7], by strengthening a result of Porschen et al. [26], we are able to show that the constructed instances of NAESAT are always satisfiable and that a satisfying NAE-truth assignment can be computed in linear time.

In [21, Section 8], we provide three reduction rules that turn a good instance I into an equivalent smaller good instance I' . Let $I^* = \langle G^*, \mathcal{S}^*, m^* \rangle$ be the good instance, equivalent to I , produced by applying a maximal sequence of reduction rules to I . We say that I^* is *irreducible*. The following lemma will prove useful.

► **Lemma 22.** *For each unmarked vertex $v \in V(G^*)$, it holds that: (i) v has degree 3, (ii) $m^*(v) = 4$, and (iii) there exist tuples $t_1, t_2 \in S^*(e)$ such that the embedding pair of t_1 and of t_2 at v are $(\text{?}, 1)$ and $(\text{!}, 1)$, respectively, for each edge e incident to v .*

Our next and final tool is the following, quite surprising, result.

► **Lemma 23.** *Any irreducible good instance I^* is a positive instance.*

Theorem 21 immediately follows from Lemma 23. We conclude the section by providing a sketch of the proof of Lemma 23. A detailed proof can be found in [21, Section 8].

Outline of the proof of Lemma 23. If a vertex is marked then any combination of tuples will satisfy m^* at it. So the proof is mainly concerned with unmarked vertices. By Lemma 22, edges where both endpoints are unmarked have one of the following tuple sets: $S_A = \{(\uparrow, 1, \uparrow, 1), (\downarrow, 1, \downarrow, 1)\}$, $S_B = \{(\downarrow, 1, \uparrow, 1), (\uparrow, 1, \downarrow, 1)\}$, or $S_A \cup S_B$. In the last case we arbitrarily remove either S_A or S_B . Taking advantage of the structure of irreducible instances, the problem of solving I^* is reduced in linear time to the one of testing the NAE-satisfiability of a CNF-formula ϕ in which every variable occurs in at most two clauses. Each edge incident to an unmarked vertex has the two possible embedding pairs $(\uparrow, 1)$ or $(\downarrow, 1)$ at the vertex. We create a variable for each incidence between an edge and an unmarked vertex. For each edge with two unmarked endpoints, we introduce an *edge clause* to ensure that the embedding pairs for each endpoint are selected in a consistent way. Consider an unmarked vertex v and assume, for simplicity of description, that its three incident edges have the same orientation at v . A selection of embedding pairs for the edges incident to v will not satisfy $m^*(v)$ if and only if all such pairs coincide. Therefore, we can introduce a *vertex clause* to model such constraint as a NAESAT clause that is the disjunction of the three boolean variables for the endpoints of the edges incident to v . The NAE-formula ϕ has the property that each variable occurs in at most two clauses. Moreover, the variable-clause graph G_ϕ of ϕ contains no connected component that is isomorphic to a simple cycle, since vertex clauses have degree 3. In [21, Section 7], we prove that such instances are always NAE-satisfiable and provide a linear-time algorithm to construct a NAE-truth assignment for such formulas. This proves that I^* is always a positive instance.

7 Conclusions

In this paper, we studied the complexity of the k -MODALITY problem, with special emphasis on $k = 4$. We provided complexity, algorithmic, and combinatorial results. Our main algorithmic contribution for $k = 4$ and $\Delta \leq 6$ leverages an elegant connection with the NAE-satisfiability of special CNF formulas, whose study allowed us to strengthen a result in [26]. Moreover, we showed notable applications of the previous results to some new interesting embedding problems for clustered networks, some of which solve open problems in this area [3, 18].

References

- 1 Patrizio Angelini, Giordano Da Lozzo, Marco Di Bartolomeo, Valentino Di Donato, Maurizio Patrignani, Vincenzo Roselli, and Ioannis G. Tollis. Algorithms and Bounds for L-Drawings of Directed Graphs. *Int. J. of Foundations of Computer Science*, 29(04):461–480, 2018. doi:10.1142/S0129054118410010.
- 2 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Ignaz Rutter. Beyond Level Planarity. In Yifan Hu and Martin Nöhlenburg, editors, *GD '16*, volume 9801 of *LNCS*, pages 482–495. Springer, 2016. doi:10.1007/978-3-319-50106-2_37.
- 3 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Ignaz Rutter. Intersection-Link Representations of Graphs. *J. Graph Algorithms Appl.*, 21(4):731–755, 2017. doi:10.7155/jgaa.00437.
- 4 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, and Vincenzo Roselli. The importance of being proper: (In clustered-level planarity and T-level planarity). *Theor. Comput. Sci.*, 571:1–9, 2015.
- 5 Patrizio Angelini, Peter Eades, Seok-Hee Hong, Karsten Klein, Stephen G. Kobourov, Giuseppe Liotta, Alfredo Navarra, and Alessandra Tappini. Turning Cliques into Paths to Achieve Planarity. In Therese C. Biedl and Andreas Kerren, editors, *GD 2018*, volume 11282 of *LNCS*, pages 67–74. Springer, 2018. doi:10.1007/978-3-030-04414-5_5.

- 6 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Valentino Di Donato, Philipp Kindermann, Günter Rote, and Ignaz Rutter. Windrose Planarity: Embedding Graphs with Direction-Constrained Edges. *ACM Trans. Algorithms*, 14(4):54:1–54:24, September 2018. doi:10.1145/3239561.
- 7 Christian Bachmaier, Franz-Josef Brandenburg, and Michael Forster. Radial Level Planarity Testing and Embedding in Linear Time. *J. Graph Algorithms Appl.*, 9(1):53–97, 2005. URL: <http://jgaa.info/accepted/2005/BachmaierBrandenburgForster2005.9.1.pdf>, doi:10.7155/jgaa.00100.
- 8 Paola Bertolazzi, Giuseppe Di Battista, Giuseppe Liotta, and Carlo Mannino. Upward Drawings of Triconnected Digraphs. *Algorithmica*, 12(6):476–497, 1994. doi:10.1007/BF01188716.
- 9 Carla Binucci, Walter Didimo, and Francesco Giordano. Maximum upward planar subgraphs of embedded planar digraphs. *Comput. Geom.*, 41(3):230–246, 2008.
- 10 Carla Binucci, Walter Didimo, and Maurizio Patrignani. Upward and quasi-upward planarity testing of embedded mixed graphs. *Theor. Comput. Sci.*, 526:75–89, 2014. doi:10.1016/j.tcs.2014.01.015.
- 11 Kellogg S. Booth and George S. Lueker. Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.
- 12 Guido Brückner and Ignaz Rutter. Partial and Constrained Level Planarity. In Philip N. Klein, editor, *SODA '17*, pages 2000–2011. SIAM, 2017. doi:10.1137/1.9781611974782.130.
- 13 Steven Chaplick, Markus Chimani, Sabine Cornelsen, Giordano Da Lozzo, Martin Nöllenburg, Maurizio Patrignani, Ioannis G. Tollis, and Alexander Wolff. Planar L-Drawings of Directed Graphs. In Fabrizio Frati and Kwan-Liu Ma, editors, *GD '17*, volume 10692 of *LNCS*, pages 465–478. Springer, 2017. doi:10.1007/978-3-319-73915-1_36.
- 14 Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, and Maurizio Patrignani. Computing NodeTrix Representations of Clustered Graphs. *J. Graph Algorithms Appl.*, 22(2):139–176, 2018.
- 15 Giuseppe Di Battista and Enrico Nardelli. Hierarchies and planarity theory. *IEEE Trans. Systems, Man, and Cybernetics*, 18(6):1035–1046, 1988. doi:10.1109/21.23105.
- 16 Giuseppe Di Battista and Roberto Tamassia. On-Line Graph Algorithms with SPQR-Trees. In Mike Paterson, editor, *ICALP '90*, volume 443 of *LNCS*, pages 598–611. Springer, 1990. doi:10.1007/BFb0032061.
- 17 Ashim Garg and Roberto Tamassia. On the Computational Complexity of Upward and Rectilinear Planarity Testing. *SIAM J. Comput.*, 31(2):601–625, 2001. doi:10.1137/S0097539794277123.
- 18 Emilio Di Giacomo, Giuseppe Liotta, Maurizio Patrignani, and Alessandra Tappini. NodeTrix Planarity Testing with Small Clusters. In Fabrizio Frati and Kwan-Liu Ma, editors, *GD '17*, volume 10692 of *LNCS*, pages 479–491. Springer, 2017. doi:10.1007/978-3-319-73915-1_37.
- 19 Nathalie Henry, Jean-Daniel Fekete, and Michael J. McGuffin. NodeTrix: a Hybrid Visualization of Social Networks. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1302–1309, 2007. doi:10.1109/TVCG.2007.70582.
- 20 John E. Hopcroft and Robert Endre Tarjan. Efficient Planarity Testing. *J. ACM*, 21(4):549–568, 1974. doi:10.1145/321850.321852.
- 21 Juan José Besa, Giordano Da Lozzo, and Michael T. Goodrich. Computing k-Modal Embeddings of Planar Digraphs. *CoRR*, abs/1907.01630, 2019. arXiv:1907.01630.
- 22 Michael Jünger, Sebastian Leipert, and Petra Mutzel. Level Planarity Testing in Linear Time. In Sue Whitesides, editor, *GD '98*, volume 1547 of *LNCS*, pages 224–237. Springer, 1998. doi:10.1007/3-540-37623-2_17.
- 23 Boris Klemz and Günter Rote. Ordered Level Planarity, Geodesic Planarity and Bi-Monotonicity. In Fabrizio Frati and Kwan-Liu Ma, editors, *GD '17*, volume 10692 of *LNCS*, pages 440–453. Springer, 2017. doi:10.1007/978-3-319-73915-1_34.

19:16 Computing k -Modal Embeddings of Planar Digraphs

- 24 JiaWei Lu, Yuanming Zhang, Jun Xu, Gang Xiao, and Qianhui Althea Liang. Data Visualization of Web Service with Parallel Coordinates and NodeTrix. In *IEEE International Conference on Services Computing, SCC 2014, Anchorage, AK, USA, June 27 - July 2, 2014*, pages 766–773. IEEE Computer Society, 2014. doi:10.1109/SCC.2014.104.
- 25 B. M. E. Moret. Planar NAE3SAT is in P. *SIGACT News*, 19(2):51–54, June 1988. doi:10.1145/49097.49099.
- 26 Stefan Porschen, Bert Randerath, and Ewald Speckenmeyer. Linear Time Algorithms for Some Not-All-Equal Satisfiability Problems. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT '03*, volume 2919 of *LNCS*, pages 172–187. Springer, 2003. doi:10.1007/978-3-540-24605-3_14.
- 27 Xinsong Yang, Lei Shi, Madelaine Daianu, Hanghang Tong, Qingsong Liu, and Paul M. Thompson. Blockwise Human Brain Network Visual Comparison Using NodeTrix Representation. *IEEE Trans. Vis. Comput. Graph.*, 23(1):181–190, 2017. doi:10.1109/TVCG.2016.2598472.

Cost Sharing over Combinatorial Domains: Complement-Free Cost Functions and Beyond

Georgios Birmpas

Department of Computer Science, University of Oxford, UK

Department of Informatics, Athens University of Economics and Business, Greece

gebirbas@gmail.com

Evangelos Markakis

Department of Informatics, Athens University of Economics and Business, Greece

markakis@aueb.gr

Guido Schäfer

Networks and Optimization group, Centrum Wiskunde & Informatica (CWI), The Netherlands

Dept. of Econometrics and Operations Research, Vrije Universiteit Amsterdam, The Netherlands

g.schaefer@cwi.nl

Abstract

We study mechanism design for combinatorial cost sharing models. Imagine that multiple items or services are available to be shared among a set of interested agents. The outcome of a mechanism in this setting consists of an assignment, determining for each item the set of players who are granted service, together with respective payments. Although there are several works studying specialized versions of such problems, there has been almost no progress for general combinatorial cost sharing domains until recently [7]. Still, many questions about the interplay between strategyproofness, cost recovery and economic efficiency remain unanswered.

The main goal of our work is to further understand this interplay in terms of budget balance and social cost approximation. Towards this, we provide a refinement of cross-monotonicity (which we term *trace-monotonicity*) that is applicable to iterative mechanisms. The trace here refers to the order in which players become finalized. On top of this, we also provide two parameterizations (complementary to a certain extent) of cost functions which capture the behavior of their average cost-shares. Based on our trace-monotonicity property, we design a scheme of ascending cost sharing mechanisms which is applicable to the combinatorial cost sharing setting with symmetric submodular valuations. Using our first cost function parameterization, we identify conditions under which our mechanism is weakly group-strategyproof, $O(1)$ -budget-balanced and $O(H_n)$ -approximate with respect to the social cost. Further, we show that our mechanism is budget-balanced and H_n -approximate if both the valuations and the cost functions are symmetric submodular; given existing impossibility results, this is best possible. Finally, we consider general valuation functions and exploit our second parameterization to derive a more fine-grained analysis of the Sequential Mechanism introduced by Moulin. This mechanism is budget balanced by construction, but in general only guarantees a poor social cost approximation of n . We identify conditions under which the mechanism achieves improved social cost approximation guarantees. In particular, we derive improved mechanisms for fundamental cost sharing problems, including Vertex Cover and Set Cover.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory and mechanism design

Keywords and phrases Approximation Algorithms, Combinatorial Cost Sharing, Mechanism Design, Truthfulness

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.20

Acknowledgements Part of this work was done while the first author was an intern of the Networks and Optimization group at Centrum Wiskunde & Informatica. The first author was also partially supported by the ERC Advanced Grant 321171 (ALGAME).



© Georgios Birmpas, Evangelos Markakis, and Guido Schäfer;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 20; pp. 20:1–20:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

How to share the cost of a common service (or public good) among a set of interested agents constitutes a fundamental problem in mechanism design that has been studied intensively for at least two decades. Several deep and significant advancements have been achieved throughout this period, notably also combining classical mechanism design objectives (such as incentive compatibility, economic efficiency, etc.) with theoretical computer science objectives (such as approximability and computational efficiency).

However, in the vast majority of the cost sharing models that have been proposed and analyzed in the literature, it is assumed that the mechanism designer is offering a single service and that each agent has a private value describing the willingness to pay for the service. At the same time, there is also a publicly known cost function which describes the total cost for offering the service to each possible subset of agents. Said differently, this results in a single-parameter mechanism design problem, where the goal is to select a subset of the players that will be granted service, subject to covering the cost and achieving an economically efficient outcome.

Although significant progress has been made for such single-parameter domains, moving towards more general *combinatorial domains* has been almost elusive so far. Imagine that there are multiple goods to be shared among the agents who now have more complex valuation functions, expressing their willingness to pay for different subsets (or bundles) of goods. The cost function now depends on the subsets of agents sharing each of the items. An outcome of a mechanism under this setting, consists of an allocation, which specifies for each agent the goods for which she is granted service, together with a payment scheme.

The desired properties in designing a cost-sharing mechanism (be it combinatorial or not) are three-fold: (i) *group-strategyproofness*: we would like resistance to misreporting preferences by individual agents or coalitions, (ii) *budget-balance*: the payments of the players should cover the incurred cost, (iii) *economic efficiency*: the allocation should maximize a measure of social efficiency. The fundamental results in [10, 16] rule out the possibility that all three properties can be achieved. As a result, if we insist on any variant of strategyproofness, we are forced to settle with approximate notions of at least one of the other two criteria. In this context, approximate budget balance means that the mechanism may overcharge the agents, but not by too much. In terms of efficiency, considering a social cost objective instead of the classical social welfare objective (definitions are given in Section 2) seems more amenable for multiplicative approximation guarantees.

These adapted objectives have been investigated thoroughly for single-parameter problems, especially for cost-sharing variants of well-known optimization problems. In the context of more general combinatorial cost-sharing mechanisms, a restricted model with multiple levels of service was first studied in [13]. Ever since, for almost a decade, there was no additional progress along these lines. It was only recently that a step forward was made by Dobzinski and Ovadia [7]. In their work, they introduce a combinatorial cost-sharing model and derive the first mechanisms guaranteeing good budget balance and social cost approximation guarantees for different classes of valuation and cost functions. As already pointed out in [7], however, several important questions concerning our understanding of the approximability of these objectives remain open and deserve further study. This constitutes the starting point of our investigations reported in this work.

Our Contributions. We make further advancements on the design and analysis of mechanisms for combinatorial cost-sharing models. To begin with, the analysis of the mechanisms we study asks for new conceptual ideas which might be interesting on their own:

- We first provide a refinement of the well-known notion of *cross-monotonic* cost sharing functions, which is key in the intensively studied class of Moulin-Shenker mechanisms [15] for the single-parameter domain. We introduce the notion of *trace-monotonic* cost sharing functions which is applicable for mechanisms that proceed iteratively and evict agents one-by-one. Trace-monotonicity formalizes the fact that the cost-shares observed by a player for an item do not decrease throughout the *course* of the mechanism. That is, these cost shares may depend on the specific order (or *trace* as we will call it) in which the mechanism considers the agents.
- We identify two different and (to some extent) complementary parameterizations of the cost functions. Intuitively, these parameters measure the “variance” of the average cost-share $c(S)/|S|$, over all agent sets S . We introduce two such notions, which we term α -average decreasing and α -average min-bounded (see Definition 5 and Definition 15, respectively). We note that for every cost function, there exist respective values of α (possibly different for each definition) for which these properties are satisfied. These definitions provide an alternative way to classify cost functions and their respective approximation guarantees in terms of budget balance and social cost.

Using the above ideas, in Section 3, we derive a scheme for ascending cost sharing mechanisms, which can be seen as a (non-trivial) adaptation of the Moulin-Shenker mechanisms from the binary accept/reject setting to combinatorial cost sharing. Our notion of trace-monotonic cost shares plays a crucial role here. We show that our proposed mechanism is applicable for any non-decreasing cost function and for symmetric submodular valuations (i.e., submodular functions whose value depends only on the cardinality of the set).

By exploiting the first parameterization of α -average decreasing cost functions, our main result of Section 3 is that for $\alpha = O(1)$, our mechanism is polynomial-time, weakly group-strategyproof, $O(1)$ -budget-balanced and $O(H_n)$ -approximate with respect to social cost, where n is the number of agents.¹ As a consequence, if both the valuation and cost functions are symmetric submodular ($\alpha = 1$), the mechanism is budget-balanced and H_n -approximate. This is best possible even for a single item, as there exist corresponding inapproximability results by Dobzinski et al. [6]. Prior to our work, the best known mechanism for symmetric submodular valuation and cost functions is H_n -budget balanced and H_n -approximate [7]. We anticipate that further extensions and generalizations might be feasible through our framework and this type of ascending mechanisms.

In Section 4, we exploit our second parameterization of α -average min-bounded cost functions, and provide results for general valuation functions. As it turns out, our parameterization enables us to obtain a more fine-grained analysis of the Sequential Mechanism introduced by Moulin [14]. This mechanism is budget-balanced by construction, but in general only guarantees a poor social cost approximation of factor n . We show that for α -average min-bounded cost functions with $\alpha = O(1)$, the Sequential Mechanism is budget balanced and H_n -approximate with respect to social cost. Interestingly, this result does not even require monotonicity of the valuation functions. In addition, we can push our results even a bit further by introducing a refinement of this class of cost functions (see Definition 20) for which we show that the Sequential Mechanism is $O(1)$ -approximate. The refinement allows us to obtain improved mechanisms for several cost functions originating from combinatorial optimization problems. For example, our result implies that the Sequential Mechanism is d -approximate for certain cost-sharing variants of Vertex Cover and Set Cover, where d is the maximum degree of a node or the maximum size of a set, respectively; this improves upon existing results, even in the well-studied single-item case, when d is constant.

¹ We use H_n to denote the n -th *Harmonic number* defined as $H_n = 1 + \frac{1}{2} + \dots + \frac{1}{n}$.

In general, the two parameterizations of the cost functions introduced in this work seem to be suitable means to accurately capture the approximation guarantees of both the ascending cost-sharing mechanism of Section 3, and the Sequential Mechanism of Section 4. In fact, we have not managed to construct natural examples of cost functions which do not admit an $O(H_n)$ -approximation by neither of the mechanisms studied here. See also the discussion in Section 5, where some examples are constructed but they are rather artificial (Proposition 25). As such, these parameterizations help us to narrow down the class of cost functions which are not yet known to admit a good social cost approximation and enhance our understanding towards further progress in combinatorial cost sharing.

Related Work. For the single-item setting and with submodular cost functions, the best known group-strategyproof and budget balanced cost-sharing mechanism is arguably the Shapley value mechanism, introduced by Moulin and Shenker [14, 15]. This was also the first work that tried to quantify the efficiency loss of budget balanced cost-sharing mechanisms. Later, Feigenbaum et al. [9] showed that if one insists on truthfulness, there is no mechanism that achieves a finite approximation of the social welfare objective, even if one relaxes the budget balance property to cost recovery. To overcome this impossibility result, Roughgarden et al. [17] introduced the notion of *social cost* as an alternative means to quantify the efficiency of a mechanism. In the same work, they showed that the Shapley value mechanism is H_n -approximate with respect to this objective. Dobzinski et al. [6] established another impossibility result for the social cost objective, and showed that every mechanism satisfying truthfulness and cost recovery cannot achieve a social cost approximation guarantee better than $\Omega(\log n)$. The problem of deriving mechanisms with the best possible budget balance and social cost approximation guarantees for different cost functions arising from combinatorial optimization problems has been extensively studied in various works, see e.g., [2, 3, 4, 11].

Moving beyond the single-item case, Mehta et al. [13] introduced a new family of truthful mechanisms (called *acyclic mechanisms*) which apply to general demand settings of multiple identical items when players have symmetric submodular valuations. For additional works that consider the general demand setting, the reader is referred to [2, 3, 5, 14]. Birmpas et al. [1] also studied families of valuation and cost functions for the multiple item setting, under cost sharing models that are motivated by applications in participatory sensing environments.

Most related to our work is the recent work by Dobzinski and Ovadia [7]. To the best of our knowledge, this is the only prior work that considers a more general approach for combinatorial cost sharing. They studied a multi-parameter setting and proposed a new VCG-based mechanism. Basically, the idea is to run a VCG mechanism with respect to a modified objective function which is defined as the sum of the player valuations minus a potential. Intuitively, the latter ensures that the payments computed by the mechanism cover the actual cost. They showed that this mechanism is strategyproof and H_n -approximate with respect to social cost. They also identified several classes of valuation and cost functions for which the mechanism is H_n -budget balanced. In particular, this is the case if the valuation and cost functions are symmetric.² Additionally, they established that their mechanism is optimal with respect to the social cost approximation among all symmetric VCG-based mechanisms that always cover the cost.

² We note that their definition of symmetry for the cost function differs from the one we use here.

2 Definitions and Notation

We assume there is a set $N = \{1, 2, \dots, n\}$ of players and a set $M = \{1, 2, \dots, m\}$ of items. Each item can be viewed as a public good or some service that can be shared by the players. Each player i has a *private valuation function* $v_i : 2^M \rightarrow \mathbb{R}_{\geq 0}$ specifying the value that she derives from each subset of items.

A *cost-sharing mechanism* takes as input the declared (possibly false) valuation functions $\vec{b} = (b_i)_{i \in N}$ of the players and outputs (i) an allocation that determines which players share each item and (ii) a payment p_i for each player i . An allocation is denoted by a tuple $A = (A_1, \dots, A_n)$, where $A_i \subseteq M$ is the set of items provided to player i . For notational convenience, we also represent an allocation $A = (A_i)_{i \in N}$ as a tuple over the items space (T_1, \dots, T_m) such that for every item $j \in M$, $T_j \subseteq N$ is the subset of players sharing item j , i.e., $T_j = \{i \in N : j \in A_i\}$.

In this paper, we consider mostly *separable* cost functions. In the *separable setting*, we assume that the overall cost of an allocation decomposes into the cost for providing each item separately. Hence, every item j is associated to a known cost function $c_j : 2^N \rightarrow \mathbb{R}_{\geq 0}$, which specifies for each set of players $T \subseteq N$, the cost $c_j(T)$ of providing item j to the players in T . Thus, the total cost of an allocation A is defined as $C(A) = \sum_{j \in M} c_j(T_j)$. In Section 4.3, we also consider the *non-separable setting*, where we are given a more general cost function $C : (2^M)^n \rightarrow \mathbb{R}_{\geq 0}$, specifying for every allocation $A = (A_i)_{i \in N}$ the corresponding cost $C(A)$. Non-separable functions can capture dependencies among different items.

We assume that the utility functions of the players are *quasilinear*, i.e., given an allocation $A = (A_i)_{i \in N}$ and payments $(p_i)_{i \in N}$ determined by the mechanism for valuation functions $\vec{v} = (v_i)_{i \in N}$, the utility of player i is defined as $u_i(\vec{v}) = v_i(A_i) - p_i$. All our mechanisms have *no positive transfers* (NPT), i.e., $p_i \geq 0$, and satisfy *individual rationality* (IR), i.e., $p_i \leq v_i(A_i)$.

In addition to the above, we are also interested in the following properties:

- **Weak Group-Strategyproofness (WGSP):** We insist on a stronger notion of resistance to manipulation than truthfulness: A mechanism is *weakly group-strategyproof* if there is no deviation by a coalition of players that makes all its members strictly better off. More formally, we require that for every coalition $Q \subseteq N$ of players, every profile \vec{v}_{-Q} of the other players, there is no deviation \vec{b}_Q of the players in Q such that $u_i(\vec{b}_Q, \vec{v}_{-Q}) > u_i(\vec{v}_Q, \vec{v}_{-Q})$ for every $i \in Q$, where \vec{v}_Q is the profile of the actual valuation functions of Q .
- **Budget Balance:** We are interested in mechanisms whose payments cover the allocation cost, ideally exactly. However, the latter is not always possible as it may be incompatible with the other objectives. We therefore consider an approximate budget balance notion: A mechanism is *β -budget-balanced* ($\beta \geq 1$) if for every valuation profile $\vec{v} = (v_i)_{i \in N}$, the outcome (A, p) computed by the mechanism satisfies

$$C(A) \leq \sum_{i \in N} p_i \leq \beta \cdot C(A).$$

Clearly, we want β to be as small as possible to not overcharge players too much for covering the cost. We say that the mechanism is *budget balanced* if $\beta = 1$.

- **Economic Efficiency:** Our goal is to compute outcomes that are (approximately) efficient. To this aim, we use the *social cost objective*, originally defined in [17]. Adapted to our combinatorial setting, the *social cost* of an allocation $A = (A_i)_{i \in N}$ is defined as the actual cost of the outcome plus the value missed by not serving all items to all players, i.e.,

$$\pi(A) = \sum_{j \in M} c_j(T_j) + \sum_{i \in N} [v_i(M) - v_i(S_i)].^3$$

A mechanism is said to be α -approximate with respect to the social cost objective if for every valuation profile $\vec{v} = (v_i)_{i \in N}$, the allocation A output by the mechanism satisfies $\pi(A) \leq \alpha \cdot \pi(A^*)$, where A^* is an allocation of minimum social cost.

We assume that both the valuation functions $(v_i)_{i \in N}$ and the cost functions $(c_j)_{j \in M}$ are non-decreasing (see below for formal definitions). Further, we focus on certain classes of valuation and cost functions: More specifically, we consider *submodular* and *subadditive* cost functions, both naturally modeling economies of scale. As to the valuation functions, we consider *submodular* valuation functions in Section 3 and general valuation functions in Section 4. Further, the class of *symmetric XOS* functions play a prominent role in Section 3.⁴ Below we summarize all relevant definitions (see also Lehman et al. [12]).

► **Definition 1.** Let $f : 2^U \rightarrow \mathbb{R}_{\geq 0}$ be a function defined over subsets of a universe U .

1. f is non-decreasing if $f(S) \leq f(T)$ for every $S \subseteq T \subseteq U$.
2. f is symmetric if $f(S) = f(T)$ for every $S, T \subseteq U$ with $|S| = |T|$.
3. f is submodular if $f(S \cup \{i\}) - f(S) \geq f(T \cup \{i\}) - f(T)$ for every $S \subseteq T \subseteq U$ and $i \notin S$.
4. f is XOS if there are additive functions f^1, \dots, f^k such that $f(S) = \max_{i \in [k]} f^i(S)$ for all $S \subseteq U$.
5. f is subadditive if $f(S \cup T) \leq f(S) + f(T)$ for every $S, T \subseteq U$.
6. f is symmetric XOS if it is symmetric and $f(S)/|S| \geq f(T)/|T|$ for every $S \subseteq T \subseteq U$.

Some of our mechanisms make use of cross-monotonic cost-sharing functions defined as follows:

► **Definition 2.** Let $c : 2^N \rightarrow \mathbb{R}_{\geq 0}$ be a cost function. A cost-sharing function⁵ $\chi : N \times 2^N \rightarrow \mathbb{R}_{\geq 0}$ with respect to c specifies for each subset $S \subseteq N$ and every player $i \in S$ a non-negative cost share $\chi(i, S)$ such that $\sum_{i \in S} \chi(i, S) \geq c(S)$.⁶ χ is cross-monotonic if for all $S \subseteq T \subseteq N$ and every $i \in S$, we have $\chi(i, S) \geq \chi(i, T)$.

3 An Iterative Ascending Cost Sharing Mechanism

In this section, we present our *Iterative Ascending Cost Sharing Mechanism (IACSM)* for the combinatorial cost sharing setting with symmetric submodular valuations and general cost functions. We first provide a generic description of our mechanism and identify two properties which are sufficient for our main result to go through. We then show that these properties are satisfied if the valuations are symmetric submodular.

3.1 Definition of IACSM and Two Crucial Properties

Mechanism IACSM can be viewed as a generalization of the *Moulin-Shenker* mechanism [15] to the combinatorial setting in the sense that it simulates in parallel an ascending iterative auction for each item. To our knowledge this is the first ascending price mechanism for the combinatorial setting which is not VCG-based and as we will describe below, this adaptation is not straightforward since there are several obstacles we need to overcome. A description of our mechanism IACSM is given in Algorithm 1.

⁴ It is not hard to verify that these functions can equivalently be defined as stated in Definition 1 (see also [8]).

⁵ We stress here that we allow cost-sharing functions to overcharge the actual cost $c(\cdot)$. As a result, this will lead to approximately budget balanced mechanisms.

⁶ For notational convenience, we define $\chi(i, S) = \infty$ for $i \notin S$.

■ **Algorithm 1** Iterative Ascending Cost Sharing Mechanism (IACSM).

Input: Declared valuation functions $(b_i)_{i \in N}$.
Output: Allocation $A = (A_i)_{i \in N}$ and payments $p = (p_i)_{i \in N}$.

- 1 **Initialization:** Let $X = N$ be the set of active players and define $T_j = N$ for every item $j \in M$.
- 2 **while** $X \neq \emptyset$ **do**
- 3 Compute an *optimal bundle* A_i for every player $i \in X$:

$$A_i \in \arg \max_{S \subseteq M} \{b_i(S) - p_i(S)\}, \quad \text{where} \quad p_i(S) = \sum_{j \in S} \chi_j(T_j) \quad (1)$$

(If there are several optimal bundles, resolve ties as described within Section 3.1.)
- 4 Let $i^* \in X$ be a player such that $|A_{i^*}| \leq |A_i|$ for every $i \in X$.
- 5 Assign the items in A_{i^*} to player i^* and remove player i^* from X .
- 6 For every item $j \in M \setminus A_{i^*}$, set $T_j = T_j \setminus \{i^*\}$, and update the cost shares $\chi_j(T_j)$.
- 7 **return** $A = (A_i)_{i \in N}$ and $p = (p_i)_{i \in N}$, where $p_i = \sum_{j \in A_i} \chi_j(T_j)$.

The mechanism maintains a set of *active* players X and for each item $j \in M$ a set of players T_j who are *tentatively* assigned to j . Initially, each player is active and tentatively assigned to all the items, i.e., $X = N$ and $T_j = N$ for all $j \in M$. The mechanism then proceeds in iterations. In each iteration, each item j is offered to each active player $i \in X$ at a price that only depends on the set of tentatively assigned players T_j . For this, we use a player-independent *cost sharing function* $\chi_j(\cdot, T_j)$ for every item j , and since we require that $\chi_j(i, T_j) = \chi_j(k, T_j)$ for every $i, k \in T_j$, we will simply denote by $\chi_j(T_j)$ the cost share that each player $i \in T_j$ tentatively assigned to j has to pay. Based on these cost shares, every active player $i \in X$ computes an *optimal bundle* A_i with respect to the payments $p_i(\cdot)$, as defined in Equation (1). If there are ties, we resolve them according to the following *tie-breaking rule*: if there are several optimal bundles, then player i chooses one of maximum size. If there are multiple optimal bundles of maximum size k , then she chooses the bundle consisting of the k cheapest items (where ties between equal cost share items are resolved consistently, say by index of the items).

After determining the optimal bundle for each active player, the mechanism then chooses an active player i^* whose optimal bundle has minimum size. Again, we break ties consistently, say by index of the players. The items in A_{i^*} are assigned to player i^* and i^* becomes inactive. Finally, for every item j that does not belong to the optimal bundle A_{i^*} , i^* is removed from the tentative set T_j . The mechanism terminates when all players are inactive.

We next identify two crucial properties that our mechanism has to satisfy for our main result to go through. To formalize these properties, we introduce first some more notation.

Trace of IACSM. Note that the execution of our mechanism IACSM on an instance of the problem induces an order $\tau = (\tau_1, \dots, \tau_n)$ on the players. Without loss of generality, we may assume that the players are renamed such that $\tau = (1, \dots, n)$, i.e., player i becomes inactive in iteration i ; however, we emphasize that this order is determined by the run of our mechanism.

The order $\tau = (1, \dots, n)$ together with the final bundle A_i assigned to each player i at the end of iteration i induces an order of player withdrawals for each item j . More precisely, for every $j \in M$ we let τ_j be the subsequence of τ consisting only of the players who withdrew from item j (at the end of the iteration when they became inactive). We refer to τ_j as the *trace of item j* . Recall that initially $T_j = N$ and hence all players are tentatively assigned to j . The length of τ_j can vary from 0, when nobody withdraws from item j and τ_j is the null sequence, all the way to n , when everybody withdraws from j and $\tau_j = \tau$. Given a trace τ_j

in the form $\tau_j = (i_1, i_2, \dots, i_\ell)$ and $k \in \{0, 1, \dots, |\tau_j|\}$, let $R_j^k = N \setminus \{i_1, i_2, \dots, i_k\}$; define $R_j^0 = N$. Note that the set R_j^k is precisely the set of players tentatively assigned to j after k players have withdrawn from j during the execution of the mechanism. We note that the notion of trace is valid also for any other iterative mechanism where the assignment of one player becomes finalized at each iteration, e.g., [13].

Trace-monotonic cost sharing functions. We introduce a new property of cost sharing functions which will turn out to be crucial below. Intuitively, it is a refinement of the standard cross-monotonicity property which has to hold only for *certain* subsets of players encountered by the mechanism, namely for the sets $\{R_j^k\}$. More precisely, given a trace τ_j for an item $j \in M$, we say that the cost sharing function χ_j is *cross-monotonic with respect to τ_j* (or, *trace-monotonic* for short), if for every $k \in \{0, \dots, |\tau_j| - 1\}$, we have $\chi_j(R_j^k) \leq \chi_j(R_j^{k+1})$. Note that this ensures that the cost share of item j (weakly) increases during the execution of the mechanism, as we consider the sequence of sets $R_j^0 \supset R_j^1 \supset \dots \supset R_j^{|\tau_j|}$. A subtle point here is that the definition of the cost share $\chi_j(R_j^k)$ may not only depend on the set of players R_j^k , but also on the trace τ_j specifying how the set R_j^k has been reached by the mechanism.⁷ It will become clear below that this additional flexibility enables us to implement our mechanism for *arbitrary* cost functions.

Properties (P1) and (P2). Our first property is rather intuitive: An item j needs to be offered to all active players at the same price and this price can only increase in subsequent iterations. In particular, this ensures that if at the end of iteration i , player i withdraws from an item $j \in M \setminus A_i$, then the price of j for the remaining players in $T_j \setminus \{i\}$ does not decrease. This is crucial to achieve strategyproofness, and it is captured precisely by trace-monotonic cost sharing functions.⁸

(P1) For each item $j \in M$ the cost sharing function χ_j is trace monotonic for every trace τ_j .

The first property alone is not sufficient to ensure that our mechanism IACSM is weakly group-strategyproof (or even strategyproof). Additionally, we need to enforce the following refinement property on the final bundles assigned to the players. We prove below that Property (P2) is satisfied for symmetric submodular valuation functions.

(P2) The final bundles $(A_i)_{i \in N}$ assigned to the players satisfy the following *refinement property*: $A_i \subseteq A_{i+1}$ for every $i \in \{1, \dots, n-1\}$.

Feasibility of (P1) and (P2). We next define the cost sharing function that we use. The intuition is as follows: Suppose that $S = T_j$ is the set of players who are tentatively allocated to item j at the beginning of iteration i for some $i \in [n]$. Ideally, we would like to charge the average cost $c_j(S)/|S|$ to each player in S , but we cannot simply do this because the average cost might decrease with respect to the previous iteration, and this will destroy Property (P1). Given our new notion of trace-monotonicity, we can resolve this by defining the cost share of item j as the maximum average cost over all player sets which were tentatively allocated to j so far.

⁷ Notationally, we would have to write here $\chi_j^{\tau_j}$ to indicate the dependency on τ_j . However, in the analysis we focus on a fixed trace produced by an execution of the mechanism and omit the explicit reference to it for notational convenience.

⁸ Note that we have to require trace-monotonicity with respect to an *arbitrary* trace of item j here, because we cannot control the trace τ_j that will be realized by IACSM.

More formally, let τ_j be the trace of item j induced by IACSM when executed on a given instance. Let S be the set of players tentatively assigned to item j at the beginning of iteration i , and fix k such that $R_j^k = S$ (by the definition of our mechanism, such a k must exist and $k \leq i - 1$). We define

$$\chi_j(S) = \max_{\ell \in \{0, \dots, k\}} \frac{c_j(R_j^\ell)}{|n - \ell|}. \quad (2)$$

Note that by using this definition we may end up overcharging the actual cost $c_j(S)$ of item j in the sense that $|S| \cdot \chi_j(S) > c_j(S)$. As we show in Section 3.2, the budget balance and social cost approximation guarantees depend on the magnitude by which we might overcharge.

It is now trivial to show that Property (P1) holds.

► **Lemma 3.** *Consider some item $j \in M$ and let $c_j : 2^N \rightarrow \mathbb{R}_{\geq 0}$ be an arbitrary cost function. Let τ_j be an arbitrary trace of j . The cost sharing function χ_j defined in (2) is trace-monotonic.*

We turn to Property (P2). In general, it seems difficult to guarantee (P2), but it is not hard to see that it holds if the valuation functions are symmetric submodular.

► **Lemma 4.** *Suppose the valuation functions are symmetric submodular. Then $A_i \subseteq A_{i+1}$ for every $i \in \{1, \dots, n - 1\}$.*

3.2 Main result for IACSM

In order to state our main result of this section, we need to introduce a crucial parameter that determines the budget balance and social cost approximation guarantees of our mechanism.

► **Definition 5.** *A cost function $c : 2^N \rightarrow \mathbb{R}_{\geq 0}$ is α -average-decreasing, for some $\alpha \geq 1$, if for every $S \subseteq T \subseteq N$, $\alpha \cdot \frac{c(S)}{|S|} \geq \frac{c(T)}{|T|}$.*

Note that for every cost function c there exists some $\alpha \geq 1$ such that c is α -average decreasing. However, here we are particularly interested in α -average decreasing cost functions for which the parameter α is small, as can be seen by Theorem 6 below. Average decreasing functions with small values of α arise naturally in the domains of digital goods and public goods. For digital goods the cost of serving a non-empty set of customers is typically assumed to be constant because there is a cost for producing the good and then it can be shared with no additional cost (hence the definition is satisfied with $\alpha = 1$). The same is applicable for some public good models. Note also that *symmetric XOS cost functions* (see Definition 1) are average-decreasing (i.e., $\alpha = 1$).

The following is the main result of this section.

► **Theorem 6.** *Suppose the valuation functions are symmetric submodular and the cost functions are α -average decreasing, for some $\alpha \geq 1$. Then the mechanism IACSM runs in polynomial time, satisfies IR, NPT, WGSP and is α -budget balanced and $2\alpha^3 H_n$ -approximate.*

Symmetric submodular cost functions are average decreasing (i.e., $\alpha = 1$) since they are a subclass of symmetric XOS functions. As a consequence, we obtain the following corollary from Theorem 6 (with an additional improvement on the social cost approximation).

► **Corollary 7.** *Suppose the valuation functions and the cost functions are symmetric submodular. Then the mechanism IACSM runs in polynomial time, satisfies IR, NPT, WGSP and is budget balanced and H_n -approximate.*

Note that the approximation factor of H_n for symmetric submodular functions is tight: The impossibility result of Dobzinski et al. [6] for a single public good implies that achieving a better approximation ratio is impossible, even in the single-item case ($m = 1$).

Finally we point out that α -average-decreasing functions are subadditive when $\alpha = 1$, while this is not necessarily true for $\alpha > 1$.

► **Lemma 8.** *Let $c(\cdot)$ be an α -average-decreasing cost function where $\alpha = 1$. Then $c(\cdot)$ is subadditive and in addition, not necessarily symmetric, or submodular. In case $c(\cdot)$ is α -average-decreasing with $\alpha > 1$, then $c(\cdot)$ is not necessarily subadditive.*

3.3 Proof of Social Cost Approximation

Due to lack of space, we will only establish the social cost approximation stated in Theorem 6. In this section, we will show that our mechanism IACSM is $2\alpha^3 H_n$ -approximate with respect to the social cost objective for symmetric submodular valuation functions.

Let $A = (A_i)_{i \in N}$ be the allocation computed by the mechanism, where $A_i \subseteq M$ is the subset of items that player i receives. As before, without loss of generality we assume that the player order induced by IACSM is $\tau = (1, \dots, n)$. Recall that for every item $j \in M$, $T_j = \{i \in N : j \in A_i\}$ is the final set of players that receive item j . We also use T_j^i to refer to the subset of players who are allocated to item j at the beginning of iteration i . Clearly, $T_j^i \supseteq T_j$ for every player i and item j .

We first state some simple lemmas which will be helpful to establish the social cost approximation guarantee.

► **Lemma 9.** *Fix an item $j \in M$ and let i be the first player in τ such that $j \in A_i$. Then $T_j = \{i, \dots, n\}$.*

► **Lemma 10.** *Consider player i who becomes inactive in iteration i . We have*

$$v_i(A_i) - \sum_{j \in A_i} \chi_j(T_j) \geq v_i(S) - \sum_{j \in S} \chi_j(T_j) \quad \forall S \subseteq M.$$

Proof. In iteration i , the final bundle A_i is chosen as the set of items maximizing the utility of player i with respect to the current cost shares, i.e.,

$$v_i(A_i) - \sum_{j \in A_i} \chi_j(T_j^i) \geq v_i(S) - \sum_{j \in S} \chi_j(T_j^i) \quad \forall S \subseteq M. \quad (3)$$

Recall that T_j^i is the set of players that are allocated to item j in iteration i . Note that by Lemma 9, $T_j^i = T_j$ for every $j \in A_i$. Further, $T_j^i \supseteq T_j$ for every $j \in M \setminus A_i$ as additional players might withdraw from j in subsequent iterations. Note that the final set T_j is reached from T_j^i by following the trace τ_j of item j . The claim now follows from the trace-monotonicity of χ_j (Property (P1)). ◀

► **Lemma 11.** *Consider player i who becomes inactive in iteration i . For every item $j \in M$,*

$$\chi_j(T_j^i) \leq \alpha \frac{c_j(\{i, \dots, n\})}{n - i + 1}.$$

► **Lemma 12.** *Let c be an α -average decreasing cost function. Let $S, T \subseteq N$ be arbitrary subsets with $|S| \leq |T|$. Then $c(S) \leq 2\alpha c(T)$.*

We are now ready to prove the approximation guarantee.

► **Lemma 13.** *Mechanisms IACSM is $2\alpha^3 H_n$ -approximate.*

Proof. Let $A^* = (A_1^*, \dots, A_n^*)$ be an optimal allocation and let T_j^* be the respective set of players that receive item j in A^* . We have

$$\begin{aligned} \pi(A) &= \sum_{i \in N} (v_i(M) - v_i(A_i)) + \sum_{j \in M} c_j(T_j) \\ &\leq \sum_{i \in N} v_i(M) - \sum_{i \in N} \left(v_i(A_i) - \sum_{j \in A_i} \chi_j(T_j) \right) \\ &\leq \sum_{i \in N} v_i(M) - \sum_{i \in N} \left(v_i(A_i^*) - \sum_{j \in A_i^*} \chi_j(T_j^i) \right) \\ &= \sum_{i \in N} (v_i(M) - v_i(A_i^*)) + \sum_{i \in N} \sum_{j \in A_i^*} \chi_j(T_j^i), \end{aligned}$$

where the first inequality holds because χ_j is α -budget balanced and the second inequality follows from Equation (3) in the proof of Lemma 10.

The proof follows if we can show that

$$\sum_{i \in N} \sum_{j \in A_i^*} \chi_j(T_j^i) \leq 2\alpha^3 H_n \sum_{j \in M} c_j(T_j^*). \quad (4)$$

We use a charging argument to prove (4). Fix some item $j \in M$ and order the players in T_j^* according to the player order $\tau = (1, \dots, n)$ induced by IACSM; let $T_j^* = \{i_1, \dots, i_{k_j^*}\}$ be the ordered set with $k_j^* := |T_j^*|$. We now “tag” each player i in T_j^* with a fraction of the cost $c_j(T_j^*)$ for item j as follows: For the l th player $i = i_l$ in T_j^* with $1 \leq l \leq k_j^*$, define

$$\text{tag}_i(j) := \frac{c_j(T_j^*)}{k_j^* - l + 1}. \quad (5)$$

That is, the first player i_1 in T_j^* is tagged with $c_j(T_j^*)/k_j^*$, the second player i_2 with $c_j(T_j^*)/(k_j^* - 1)$ and so forth, and the last player $i_{k_j^*}$ is tagged with $c_j(T_j^*)$.

We first derive two lower bounds on the tagged cost:

► **Claim 14.** For every player $i \in N$ and for every item $j \in A_i^*$:

$$\text{tag}_i(j) \geq \frac{c_j(T_j^*)}{n - i + 1} \quad \text{and} \quad \text{tag}_i(j) \geq \frac{c_j(T_j^*)}{|T_j^*|}.$$

Proof. The latter bound holds by definition (5). To see that the former bound holds, observe that the k th last player ($1 \leq k \leq k_j^*$) in the ordered set T_j^* is tagged by $c_j(T_j^*)/k$. The claim now follows because there are at most $n - i$ players succeeding i in T_j^* according to the order. ◁

Note that the total tagged cost of item j satisfies

$$\sum_{i \in T_j^*} \text{tag}_i(j) = \sum_{l=1}^{k_j^*} \frac{c_j(T_j^*)}{k_j^* - l + 1} \leq H_n c_j(T_j^*). \quad (6)$$

Thus, to prove (4) it suffices to show that the total cost share sum is upper bounded by the total tagged cost, i.e.,

$$\sum_{i \in N} \sum_{j \in A_i^*} \chi_j(T_j^i) \leq 2\alpha^3 \sum_{j \in M} \sum_{i \in T_j^*} \text{tag}_i(j). \quad (7)$$

20:12 Cost Sharing over Combinatorial Domains

We show that for every i and every $j \in A_i^*$, $\chi_j(T_j^i) \leq \text{tag}_i(j)$. Summing over all $i \in N$ and $j \in A_i^*$ then proves (4). We distinguish two cases:

Case 1: $|T_j^*| \geq n - i + 1$: Let $S \subseteq T_j^*$ be a set such that $|S| = n - i + 1$. We have

$$\chi_j(T_j^i) \leq \alpha \frac{c_j(\{i, \dots, n\})}{n - i + 1} \leq 2\alpha^2 \frac{c_j(S)}{|S|} \leq 2\alpha^2 \frac{c_j(T_j^*)}{n - i + 1} \leq 2\alpha^2 \text{tag}_i(j), \quad (8)$$

where the first inequality follows from Lemma 11, the second inequality follows from Lemma 12, the third inequality holds because c_j is non-decreasing and the last inequality follows from Claim 14.

Case 2: $|T_j^*| < n - i + 1$: Let $S \supset T_j^*$ be a set such that $|S| = n - i + 1$. We have

$$\chi_j(T_j^i) \leq \alpha \frac{c_j(\{i, \dots, n\})}{n - i + 1} \leq 2\alpha^2 \frac{c_j(S)}{|S|} \leq 2\alpha^3 \frac{c_j(T_j^*)}{|T_j^*|} \leq 2\alpha^3 \text{tag}_i(j), \quad (9)$$

where the first inequality follows from Lemma 11, the second inequality follows from Lemma 12, the third inequality holds because c_j is α -average-decreasing and the last inequality follows from Claim 14. This concludes the proof. \blacktriangleleft

4 Mechanisms for General Valuations and Subadditive Cost Functions

In this section, we move away from symmetric submodular valuation functions and derive results for more general functions. In particular, we investigate the performance of the *Sequential Mechanism* [14] for general valuations and subadditive cost functions. Although for arbitrary subadditive cost functions this mechanism does not provide favorable approximation guarantees, we identify conditions on the cost functions under which it achieves significantly better approximation factors. This is based on considering a different parameterization of cost functions with regard to their average cost shares.

4.1 The Sequential Mechanism

The *Sequential Mechanism* (SM) was introduced by Moulin [14] and was also studied in [7]. A description of the mechanism SM is given in Algorithm 2. We note that this mechanism is applicable both to separable and non-separable cost functions. Here, we first focus on separable cost functions, and in Section 4.3, we consider generalizations to the non-separable setting.

It is trivial to see that SM is budget-balanced and it is also known that it is WGSP [7]. However, for arbitrary monotone subadditive cost functions, the mechanism achieves a (poor) social cost approximation guarantee of n (see [7]). Despite this, we show that SM has better guarantees under certain conditions. Namely, we identify a crucial parameter of each cost function c_j with $j \in M$ that allows us to quantify this improvement. The parameterization introduced here is different from the one used in Section 3 and it compares the average cost of a set $T \subseteq N$ with the minimum standalone cost of a player in T .

► Definition 15. A cost function $c : 2^N \rightarrow \mathbb{R}_{\geq 0}$ is α -average min-bounded, for some $\alpha \geq 1$, if for every set $T \subseteq N$, we have $\alpha \cdot \frac{c(T)}{|T|} \geq c_{\min}(T)$, where $c_{\min}(T) = \min_{j \in T} c(\{j\})$.

Definition 15 may look contrived at first glance and we thus provide some more intuition on how we arrived at this parameterization. Given that IACSM performs well for α -average decreasing functions and small values of α , as we established in Section 3, it is natural to

■ **Algorithm 2** Sequential Mechanism (SM).

Input: Declared valuation functions $(b_i)_{i \in N}$.
Output: Allocation $A = (A_i)_{i \in N}$ and payments $p = (p_i)_{i \in N}$.

- 1 **Initialization:** Fix an order on the set of players $N = \{1, \dots, n\}$.
- 2 **for** $i = 1, \dots, n$ **do**
- 3 Compute an *optimal bundle* A_i for player i :

$$A_i \in \arg \max_{S \subseteq M} \{b_i(S) - p_i(S)\}, \quad \text{where}$$

$$p_i(S) = C(A_1, \dots, A_{i-1}, S, \emptyset, \dots, \emptyset) - C(A_1, \dots, A_{i-1}, \emptyset, \dots, \emptyset).$$

(If there are multiple optimal bundles, choose the lexicographically smallest one.)
- 4 **return** $A = (A_i)_{i \in N}$ and $p = (p_i)_{i \in N}$, where $p_i = p_i(A_i)$.

focus on the complement of this class. For example, fix $\alpha = 1$ for now. Then the exact complement is not easy to characterize because it involves two existential quantifiers. We therefore consider a subset of this complement (with only one existential quantifier) by demanding that for every T , there exists $S \subseteq T$ such that $c(S)/|S| < c(T)/|T|$. It is not hard to verify that this definition is equivalent to the class of 1-average min-bounded functions. For larger values of α , we can see that α -average-min-bounded functions still capture a chunk of the complement of α -average-decreasing functions. Thus, a positive result for α -average-min-bounded functions narrows down on the cost functions that are not yet known to admit good approximation guarantees.

Note that for every cost function we can find an $\alpha \geq 1$ such that it is α -average min-bounded. As the next theorem reveals, the Sequential Mechanism attains a favorable performance for small values of α .

► **Theorem 16.** *Suppose we have general valuation functions and for each item $j \in M$, the cost function $c_j : 2^N \rightarrow \mathbb{R}_{\geq 0}$ is non-decreasing, subadditive, and α -average min-bounded for some $\alpha \geq 1$. Then the Sequential Mechanism satisfies IR, NPT, WGSP, and is budget balanced and $\alpha \cdot H_n$ -approximate.*

For the proof of Theorem 16, we will use the following proposition:

► **Proposition 17.** *If $c : 2^N \rightarrow \mathbb{R}_{\geq 0}$ is non-decreasing and α -average min-bounded, then $\sum_{i \in T} c(\{i\}) \leq \alpha H_{|T|} \cdot c(T)$ for every $T \subseteq N$.*

Proof of Theorem 16. We only need to prove that SM is αH_n -approximate. All the other properties have been established in [7, 14]. Let $A = (A_i)_{i \in N}$ be the allocation output by the mechanism and let $A^* = (A_i^*)_{i \in N}$ be an optimal allocation. Further, let T_j^* be the respective set of players that receive item j in A^* . To simplify notation in the analysis, we also let $A_{<i}$ denote the tuple $(A_1, \dots, A_{i-1}, \emptyset, \dots, \emptyset)$. Define now the incremental cost of a player i for a bundle $S \subseteq M$, with respect to the allocation constructed by the Sequential Mechanism before i 's turn as $\Delta_i(A_{<i}, S) = C(A_1, \dots, A_{i-1}, S, \emptyset, \dots, \emptyset) - C(A_1, \dots, A_{i-1}, \emptyset, \dots, \emptyset)$.

We have

$$\begin{aligned} \pi(A) &= \sum_{i \in N} [v_i(M) - v_i(A_i)] + C(A) = \sum_{i \in N} v_i(M) - \sum_{i \in N} [v_i(A_i) - \Delta_i(A_{<i}, A_i)] \\ &\leq \sum_{i \in N} [v_i(M) - v_i(A_i^*)] + \sum_{i \in N} \Delta_i(A_{<i}, A_i^*). \end{aligned}$$

20:14 Cost Sharing over Combinatorial Domains

Note that the inequality holds because A_i was chosen as the optimal bundle for i . The next step is to prove a bound on the incremental costs in the form

$$\sum_{i \in N} \Delta_i(A_{<i}, A_i^*) \leq \beta \cdot C(A^*). \quad (10)$$

The proof follows if we can show that (10) holds for $\beta = \alpha H_n$ because we then have

$$\pi(A) \leq \sum_{i \in N} [v_i(M) - v_i(A_i^*)] + \alpha \cdot H_n C(A^*) \leq \alpha H_n \cdot \pi(A^*).$$

By exploiting the subadditivity of the cost functions c_j , we obtain

$$\Delta_i(A_{<i}, A_i^*) = C(A_{<i}, A_i^*) - C(A_{<i}) \leq C(A_{<i}) + C(A_i^*, \emptyset_{-i}) - C(A_{<i}) = \sum_{j \in A_i^*} c_j(\{i\}).$$

Summing over over all $i \in N$, and using Proposition 17, we get:

$$\sum_{i \in N} \Delta_i(A_{<i}, A_i^*) \leq \sum_{i \in N} \sum_{j \in A_i^*} c_j(\{i\}) = \sum_{j \in M} \sum_{i \in T_j^*} c_j(\{i\}) \leq \sum_{j \in M} \alpha H_{|T_j^*|} c_j(T_j^*) \leq \alpha H_n C(A^*).$$

This proves (10) and concludes the proof of Theorem 16. \blacktriangleleft

By going through the proof of Theorem 16 more carefully, we realize the following:

► **Remark 18.** For any subclass of non-decreasing, subadditive cost functions, it suffices to establish inequality (10) to prove that the Sequential Mechanism has a social cost approximation guarantee of β .

Finally we have that for $\alpha = 1$ the approximation factor is tight.

► **Proposition 19.** *Even for the single item setting, there exists a 1-average min-bounded cost function, under which the Sequential Mechanism provides an H_n -approximation.*

4.2 Improved Approximation Guarantees and Applications

We continue with a natural refinement of Definition 15 which turns out to provide even better approximation factors of the Sequential Mechanism.

► **Definition 20.** *A cost function $c : 2^N \rightarrow \mathbb{R}_{\geq 0}$ is α -average max-bounded, for some $\alpha \geq 1$, if for every set $T \subseteq N$, we have $\alpha \cdot \frac{c(T)}{|T|} \geq c_{\max}(T)$, where $c_{\max}(T) = \max_{j \in T} c(\{j\})$.*

Clearly, any function that is α -average max-bounded is also α -average min-bounded. Thus, we already have an αH_n -approximation for non-decreasing, subadditive and α -average max-bounded cost functions. Below we show that we can achieve a much better guarantee.

► **Theorem 21.** *Suppose we have general valuation functions and for each item $j \in M$, the cost function $c_j : 2^N \rightarrow \mathbb{R}_{\geq 0}$ is non-decreasing, subadditive, and α -average max-bounded for some $\alpha \geq 1$. Then the Sequential Mechanism satisfies IR, NPT, WGSP, and is budget-balanced and α -approximate.*

Example applications of combinatorial cost functions. We give some examples of combinatorial cost functions below and show that they are α -average max-bounded (possibly depending on some parameters of the combinatorial problem). In particular, by applying Theorem 21 we obtain attractive social cost approximation guarantees for these problems. For simplicity, all examples consider a single item only; but clearly, we can consider more general multiple item settings (e.g., when for each item $j \in M$, c_j captures one of the problems below).

- **Set Cover.** We are given a universe of elements U and a family $\mathcal{F} \subseteq 2^U$ of subsets of U . The players correspond to the elements of U and the cost $c(S)$ for serving a set of players $S \subseteq U$ is defined as the size of a minimum cardinality set cover for S .
- **Vertex Cover.** This is a special case of Set Cover. We are given an undirected and unweighted graph $G = (V, E)$ and the players are the edges of the graph. The cost $c(S)$ for serving a set $S \subseteq E$ of players is defined as the size of a minimum vertex cover in the subgraph induced by S .
- **Matching.** We are given an undirected and unweighted graph $G = (V, E)$ and the players correspond to the edges. The cost $c(S)$ for serving a set S of players is defined as the size of a maximum cardinality matching in the subgraph induced by S .

Using our α -average max-bounded notion, it is now easy to prove that these problems admit constant social cost approximation guarantees (under certain restrictions).

► **Theorem 22.** *The Sequential Mechanism is α -approximate for the above problems, where*

1. $\alpha = d$ for the Set Cover problem, where d is the maximum cardinality of the sets in \mathcal{F} ;
2. $\alpha = k$ for the Vertex Cover problem in graphs of maximum degree k ;
3. $\alpha = k$ for the Matching problem in bipartite graphs of maximum degree k ;
4. $\alpha = (5k + 3)/4$ for the Matching problem in general graphs of maximum degree k .

We can now compare these bounds with the existing results in the literature. For Vertex Cover, there is a mechanism that is 2-budget-balanced and $O(\log n)$ -approximate [13]. Thus, for graphs with maximum degree less than $\log n$, we obtain a better guarantee. For Set Cover, there is a mechanism that is $O(\log n)$ -budget-balanced and $O(\log n)$ -approximate [13]. Hence, we obtain an improvement if the sets in \mathcal{F} have a size that is no more than $o(\log n)$. Finally, we note that our results do not apply to the weighted versions of these problems.

4.3 Guarantees of the Sequential Mechanism for Non-Separable Cost Functions

We extend our results to non-separable cost functions. Recall that in this setting, the cost $C(A)$ of an allocation $A = (A_i)_{i \in N}$ is given by some general (not necessarily separable) cost function $C : (2^M)^n \rightarrow \mathbb{R}_{\geq 0}$. In particular, C may encode dependencies among different items.

We introduce some more notation. Given two allocations $S = (S_i)_{i \in N}$ and $T = (T_i)_{i \in N}$, we define $S \cup T$ as the componentwise union of S and T , i.e., $S \cup T = (S_1 \cup T_1, \dots, S_n \cup T_n)$. Similarly, we write $S \subseteq T$ if this relation holds componentwise, i.e., $S_i \subseteq T_i$ for every $i \in N$. Given an allocation $A = (A_i)_{i \in N}$ and a set of players $S \subseteq N$, we define $A|_S = (A_S, \emptyset_{-S})$ as the allocation in which each player $i \in S$ receives the items in A_i and all other players receive nothing. If $S = \{i\}$ is a singleton set, we also write $A|_i$ instead of $A|_{\{i\}}$. Throughout this section, we remain in the domain of non-decreasing and subadditive cost functions. In the non-separable case, a cost function $C : (2^M)^n \rightarrow \mathbb{R}_{\geq 0}$ is non-decreasing if $C(S) \leq C(T)$ for every pair of allocations S, T , with $S \subseteq T$. Also, it is subadditive if for every two allocations $S = (S_i)_{i \in N}$ and $T = (T_i)_{i \in N}$, we have $C(S \cup T) \leq C(S) + C(T)$.

We now adapt Definitions 15 and 20 to non-separable cost functions.

- **Definition 23.** Let $C : (2^M)^n \rightarrow \mathbb{R}_{\geq 0}$ be a non-separable cost function.
- C is α -average min-bounded, for some $\alpha \geq 1$, if for every allocation A and every subset $T \subseteq N$ with $|T| \geq 2$, it holds $\alpha \frac{C(A|_T)}{|T|} \geq C_{\min}(T)$, where $C_{\min}(T) = \min_{j \in T} C(A|_j)$.
 - C is α -average max-bounded, for some $\alpha \geq 1$, if for every allocation A and every subset $T \subseteq N$ with $|T| \geq 2$, it holds $\alpha \frac{C(A|_T)}{|T|} \geq C_{\max}(T)$, where $C_{\max}(T) = \max_{j \in T} C(A|_j)$.

As before, if a non-separable function is α -average max-bounded, then it is also α -average min-bounded.

We remark that it has been shown in [14, 7] that the Sequential Mechanism is weakly group-strategyproof and budget balanced for the non-separable setting. By adapting Proposition 17 for the non-separable setting and by using the same reasoning as in the proof of Theorem 16, we obtain the same approximation guarantee of αH_n as in the separable setting. Further, the improvement we obtained in Theorem 21 also goes through in this setting. We summarize these observations in the following corollary.

- **Corollary 24.** Suppose we have general valuation functions and a non-decreasing, subadditive, and α -average min-bounded cost function $C : (2^M)^n \rightarrow \mathbb{R}_{\geq 0}$. Then the Sequential Mechanism satisfies IR, NPT, WGSP, and is budget balanced and $\alpha \cdot H_n$ -approximate. Furthermore, if C is also α -average max-bounded, then the Sequential Mechanism is α -approximate.

5 Discussion

In Section 3, we proposed the mechanism IACSM, which is weakly group-strategyproof under general cost functions and symmetric submodular valuations. Moreover it is α -budget balanced and $2\alpha^3 H_n$ -approximate when we restrict the cost functions to the α -average-decreasing class. The social cost approximation guarantee further improves to H_n if the cost functions are symmetric submodular and this is best possible (due to the known lower bound for public-excludable goods [6]). It would be very interesting to explore mechanisms that go beyond symmetric submodular valuation functions. It seems that entirely new ideas are needed for this setting. It would also be interesting to extend our mechanism to non-separable cost functions. We note that separability of the costs in Section 3 is needed for IACSM only to argue that the cost share per item increases as players withdraw (with respect to the trace). One would need to investigate how to adapt the mechanism and enforce this property in the non-separable setting. Technically, this seems far from obvious and we leave a proper treatment of this issue for future work.

In Section 4, we studied the (partially) complementary class of α -average min bounded cost functions. We showed that the well-known Sequential Mechanism is budget balanced and αH_n -approximate even for general valuation functions. These results also extend to non-separable cost functions. A very natural question is whether SM is optimal in this setting and we note that the answer is not yet clear: The impossibility result of [6] holds for the public-excludable good cost function which is symmetric submodular and thus 1-average-decreasing. However, it is not hard to see that this does not fall within the α -average min bounded class for any constant α . This leads to the question of whether there exists a WGSP mechanism that breaks the $\Omega(\log(n))$ -approximation in terms of social cost for α -average-min bounded functions with small values of α .

Finally, what we also find very interesting is to identify the class of cost functions for which neither of the two mechanisms studied here perform well. Recall that, for any constant value of α , if a cost function is either α -average decreasing or α -average min-bounded, then a good performance is guaranteed. Thus, we need look at the complement of the set of

α -average decreasing functions and the set of α -average min-bounded functions for small value of α and examine whether these complements have a non-empty intersection. The following proposition shows that this intersection is indeed non-empty.

► **Proposition 25.** *Given $\alpha \geq 1$, the intersection of the complements of α -average-decreasing and α -average min-bounded functions is non-empty.*

The proof of this proposition follows by constructing a cost function that requires non-constant values of α to be captured by either of our parameterizations. Although the intersection turns out to be non-empty, the constructed cost function is rather artificial and more natural examples are elusive so far. In fact, for most of the known cost functions that have been studied in the literature, at least one of our mechanisms achieves an $O(H_n)$ -approximation. To make further progress, we believe it is important to understand better the class of functions defined by the intersection of the two complements, as it would help us to identify the missing elements for deriving mechanisms for a wider class of cost functions.

References

- 1 G. Birmpas, C. Courcoubetis, I. Giotis, and E. Markakis. Cost-Sharing Models in Participatory Sensing. In *International Symposium on Algorithmic Game Theory*, pages 43–56, 2015.
- 2 Y. Bleischwitz and F. Schoppmann. Group-Strategyproof Cost Sharing for Metric Fault Tolerant Facility Location. In *International Symposium on Algorithmic Game Theory*, pages 350–361, 2008.
- 3 J. A. Brenner and G. Schäfer. Cost Sharing Methods for Makespan and Completion Time Scheduling. In *Symposium on Theoretical Aspects of Computer Science*, pages 670–681, 2007.
- 4 S. Chawla, T. Roughgarden, and M. Sundararajan. Optimal Cost-Sharing Mechanisms for Steiner Forest Problems. In *International Workshop on Internet and Network Economics*, pages 112–123, 2006.
- 5 N. R. Devanur, M. Mihail, and V. V. Vazirani. Strategyproof cost-sharing mechanisms for set cover and facility location games. *Decision Support Systems*, 39(1):11–22, 2005.
- 6 S. Dobzinski, A. Mehta, T. Roughgarden, and M. Sundararajan. Is Shapley cost sharing optimal? *Games and Economic Behavior*, 108:130–138, 2018.
- 7 S. Dobzinski and S. Ovadia. Combinatorial Cost Sharing. In *ACM Conference on Economics and Computation*, pages 387–404, 2017.
- 8 T. Ezra, M. Feldman, T. Roughgarden, and W. Suksompong. Pricing Identical Items. *CoRR*, abs/1705.06623, 2017. [arXiv:1705.06623](https://arxiv.org/abs/1705.06623).
- 9 J. Feigenbaum, A. Krishnamurthy, R. Sami, and S. Shenker. Hardness results for multicast cost sharing. *Theoretical Computer Science*, 1-3(304):215–236, 2003.
- 10 J. Green, E. Kohlberg, and J. J. Laffont. Partial Equilibrium Approach to the Free Rider Problem. *Journal of Public Economics*, 6:375–394, 1976.
- 11 A. Gupta, Jochen Könemann, Stefano Leonardi, R. Ravi, and Guido Schäfer. Efficient cost-sharing mechanisms for prize-collecting problems. *Math. Program.*, 152(1-2):147–188, 2015.
- 12 B. Lehmann, D. Lehmann, and N. Nisan. Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55(2):270–296, 2006.
- 13 A. Mehta, T. Roughgarden, and M. Sundararajan. Beyond Moulin mechanisms. *Games and Economic Behavior*, 67(1):125–155, 2009.
- 14 H. Moulin. Incremental cost sharing: Characterization by coalition strategy-proofness. *Soc. Choice Welfare*, 16:279–320, 1999.
- 15 H. Moulin and S. Shenker. Strategyproof sharing of submodular costs: Budget balance vs efficiency. *Economic Theory*, 18:511–533, 2001.
- 16 K. Roberts. The Characterization of Implementable Choice Rules. In J. J. Laffont, editor, *Aggregation and Revelation of Preferences*. Amsterdam: North Holland, 1979.
- 17 T. Roughgarden and M. Sundararajan. Quantifying inefficiency in cost-sharing mechanisms. *Journal of the ACM*, 56(4):23:1–23:33, 2009.

Efficiently Generating Geometric Inhomogeneous and Hyperbolic Random Graphs

Thomas Bläsius

Hasso Plattner Institute, Potsdam, Germany
thomas.blaesius@hpi.de

Tobias Friedrich

Hasso Plattner Institute, Potsdam, Germany
tobias.friedrich@hpi.de

Maximilian Katzmann

Hasso Plattner Institute, Potsdam, Germany
maximilian.katzmann@hpi.de

Ulrich Meyer

Goethe University, Frankfurt, Germany
umeyer@ae.cs.uni-frankfurt.de

Manuel Penschuck

Goethe University, Frankfurt, Germany
mpenschuck@ae.cs.uni-frankfurt.de

Christopher Weyand

Hasso Plattner Institute, Potsdam, Germany
christopher.weyand@hpi.de

Abstract

Hyperbolic random graphs (HRG) and geometric inhomogeneous random graphs (GIRG) are two similar generative network models that were designed to resemble complex real world networks. In particular, they have a power-law degree distribution with controllable exponent β , and high clustering that can be controlled via the temperature T .

We present the first implementation of an efficient GIRG generator running in expected linear time. Besides varying temperatures, it also supports underlying geometries of higher dimensions. It is capable of generating graphs with ten million edges in under a second on commodity hardware. The algorithm can be adapted to HRGs. Our resulting implementation is the fastest sequential HRG generator, despite the fact that we support non-zero temperatures. Though non-zero temperatures are crucial for many applications, most existing generators are restricted to $T = 0$. We also support parallelization, although this is not the focus of this paper. Moreover, we note that our generators draw from the correct probability distribution, i.e., they involve no approximation.

Besides the generators themselves, we also provide an efficient algorithm to determine the non-trivial dependency between the average degree of the resulting graph and the input parameters of the GIRG model. This makes it possible to specify the desired expected average degree as input.

Moreover, we investigate the differences between HRGs and GIRGs, shedding new light on the nature of the relation between the two models. Although HRGs represent, in a certain sense, a special case of the GIRG model, we find that a straight-forward inclusion does not hold in practice. However, the difference is negligible for most use cases.

2012 ACM Subject Classification Theory of computation → Generating random combinatorial structures

Keywords and phrases hyperbolic random graphs, geometric inhomogeneous random graph, efficient network generation

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.21

Related Version A full version of the paper is available at [3], <http://arxiv.org/abs/1905.06706>.



© Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, Ulrich Meyer, Manuel Penschuck, and Christopher Weyand;

licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 21; pp. 21:1–21:14

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Network models play an important role in different fields of science [8]. From the perspective of network science, models can be used to explain observed behavior in the real world. To mention one example, Watts and Strogatz [26] observed that few random long-range connections suffice to guarantee a small diameter. This explains why many real-world networks exhibit the small-world property despite heavily favoring local over long-range connections. From the perspective of computer science, and specifically algorithmics, realistic random networks can provide input instances for graph algorithms. This facilitates theoretical approaches (e.g., average-case analysis), as well as extensive empirical evaluations by providing an abundance of benchmark instances, solving the pervasive scarcity of real-world instances.

There are some crucial features that make a network model useful. The generated instances have to resemble real-world networks. The model should be as simple and natural as possible to facilitate theoretical analysis, and to prevent untypical artifacts. And it should be possible to efficiently draw networks from the model. This is particularly important for the empirical analysis of model properties and for generating benchmark instances.

A model that has proven itself useful in recent years is the *hyperbolic random graph (HRG)* model [17]. HRGs are generated by drawing vertex positions uniformly at random from a disk in the hyperbolic plane. Two vertices are joined by an edge if and only if their distance lies below a certain threshold; see Section 2.2. HRGs resemble real-world networks with respect to crucial properties. Most notable are the *power-law degree distribution* [15] (i.e., the number of vertices of degree k is roughly proportional to $k^{-\beta}$ with $\beta \in (2, 3)$), the high *clustering coefficient* [15] (i.e., two vertices are more likely to be connected if they have a common neighbor), and the small diameter [11, 19]. Moreover, HRGs are accessible for theoretical analysis (see, e.g., [15, 11, 19, 4]). Finally there is a multitude of efficient generators with different emphases [2, 24, 23, 25, 21, 13, 12]; see Section 1.2 for a discussion.

Closely related to HRGs is the *geometric inhomogeneous random graph (GIRG)* model [7]. Here every vertex has a position on the d -dimensional torus and a weight following a power law. Two vertices are then connected if and only if their distance on the torus is smaller than a threshold based on the product of their weights. When using positions on the circle ($d = 1$), GIRGs approximate HRGs in the following sense: the processes of generating a HRG and a GIRG can be coupled such that it suffices to decrease and increase the average degree of the GIRG by only a constant factor to obtain a subgraph and a supergraph of the corresponding HRG, respectively. Compared to HRGs, GIRGs are potentially easier to analyze, generalize nicely to higher dimensions, and the weights allow to directly adjust the degree distribution.

Above, we described the idealized *threshold variants* of the models, where two vertices are connected if and only if their distance is small enough. Arguably more realistic are the *binomial variants*, which allow longer edges and shorter non-edges with a small probability. This is achieved with an additional parameter T , called *temperature*. For $T \rightarrow 0$, the binomial and threshold variants coincide. Many publications focus on the threshold case, as it is typically simpler. This is particularly true for generation algorithms: in the threshold variants one can ignore all vertex pairs with sufficient distance, which can be done using geometric data structures. In the binomial case, any pair of vertices could be adjacent, and the search space cannot be reduced as easily. For practical purposes, however, a non-zero temperature is crucial as real-world networks are generally assumed to have positive temperature. Moreover, from an algorithmic perspective, the threshold variants typically produce particularly well-behaved instances, while a higher temperature leads to difficult problem inputs. Thus, to obtain benchmark instances of varying difficulty, generators for the binomial variants are key.

1.1 Contribution & Outline

Based on the algorithm by Bringmann, Keusch, and Lengler [7], we provide an efficient and flexible GIRG generator. It includes the binomial case and allows higher dimensions. Its expected running time is linear. To the best of our knowledge, this is the first efficient generator for the GIRG model. Moreover, we adapt the algorithm to the HRG model, including the binomial variant. Compared to existing HRG generators (most of which only support the threshold variant), our implementation is the fastest sequential HRG generator.

A refactoring of the original GIRG algorithm [7] allows us to parallelize our generators. They do not use multiple processors as effectively as the threshold-HRG generator by Penschuck [21], which was specifically tailored towards parallelism. However, in a setting realistic for commodity hardware (8 cores, 16 threads), we still achieve comparable run times.

Our generators come as an open-source C++ library¹ with documentation, command-line interface, unit tests, micro benchmarks, and OpenMP [6] parallelization using shared memory. An integration into NetworkKit [22] is available.

Besides the efficient generators, we have three secondary contributions. (I) We provide a comprehensible description of the sampling algorithm that should make it easy to understand how the algorithm works, why it works, and how it can be implemented. Although the core idea of the algorithm is not new [7], the previous description is somewhat technical. (II) The expected average degree can be controlled via an input parameter. However, the dependence of the average degree on the actual parameter is non-trivial. In fact, given the average degree, there is no closed formula to determine the parameter. We provide a linear-time algorithm to estimate it. (III) We investigate how GIRGs and HRGs actually relate to each other by measuring how much the average degree of the GIRG has to be decreased and increased to obtain a subgraph and supergraph of the HRG, respectively.

In the following we first discuss our main contribution in the context of existing HRG generators. In Section 2, we formally define the GIRG and HRG models. Afterwards we describe the sampling algorithm in Section 3. In Section 4 we discuss implementation details, including the parameter estimation for the average degree (Section 4.1) as well as multiple performance improvements. Section 5 contains our experiments: we investigate the scaling behavior of our generator in Section 5.1, compare our HRG generator to existing ones in Section 5.2, and compare GIRGs to HRGs in Section 5.3.

1.2 Comparison with Existing Generators

We are not aware of previous GIRG implementations. Concerning HRGs, most algorithms only support the threshold case; see Table 1. The only published exceptions are the trivial quadratic algorithm [2], and an $O((n^{3/2} + m) \log n)$ algorithm [23] based on a quad-tree data structure [24]. The latter is part of NetworkKit; we call it NKQUAD. Moreover, the code for a hyperbolic embedding algorithm [5] includes an HRG generator implemented by Bringmann based on the GIRG algorithm [7]; we call it EMBEDDER in the following. EMBEDDER has been widely ignored as a high performance generator. This is because it was somewhat hidden, and it is heavily outperformed by other threshold generators. Experiments show that our generator HYPERGIRGs is much faster than NKQUAD, which is to be expected considering the asymptotic running time. Moreover, on a single processor, we outperform EMBEDDER by an order of magnitude for $T = 0$ and by a factor of 4 for higher temperatures. As EMBEDDER does not support parallelization, this speed-up increases for multiple processors.

¹ <https://github.com/chistopher/girgs>

■ **Table 1** Existing hyperbolic random graph generators. The columns show the names used throughout the paper; the conference appearance; a reference (journal if available); whether the generator supports the binomial model; and the asymptotic running time. The time bounds hold in the worst case (wc), with high probability (whp), in expectation (exp), or empirically (emp).

Name	First Published	Ref.	Binom.	Running Time
PAIRWISE	CPC'15	[2]	✓	$\Theta(n^2)$ (wc)
QUADTREE	ISAAC'15	[24]		$O((n^{3/2} + m) \log n)$ (wc)
NKQUAD	IWOCA'16	[23]	✓	$O((n^{3/2} + m) \log n)$ (wc)
NKGEN, NKOPT	HPEC'16	[25]		$O(n \log n + m)$ (emp)
EMBEDDER	ESA'16	[5]	✓	$\Theta(n + m)$ (exp)
HYPERGEN	SEA'17	[21]		$O(n \log \log n + m)$ (whp)
RHG	IPDPS'18	[13]		$\Theta(n + m)$ (exp)
SRHG	JPDC'19	[12]		$\Theta(n + m)$ (exp)
HYPERGIRGS	this paper		✓	$\Theta(n + m)$ (exp)

For the threshold variant of HRGs, there are the following generators. The quad-tree data structure mentioned above was initially used for a threshold generator (QUADTREE) [24]. It was later improved leading to the algorithm currently implemented in NetworKit (NKGEN) [25]. A later re-implementation by Penschuck [21] improves it by about a factor of 2 (NKOPT). However, the main contribution of [21] was a new generator that features sublinear memory and near optimal parallelization (HYPERGEN). Up to date, HYPERGEN was the fastest threshold-HRG generator on a single processor. Our generator, HYPERGIRGS, improves by a factor of 1.3 – 2 (depending on the parameters) but scales worse for more processors. Finally, Funke et al. [13] provide a generator designed for a distributed setting to generate enormous instances (RHG). Its run time was later further reduced (SRHG) [12].

2 Models

2.1 Geometric Inhomogeneous Random Graphs

GIRGs [7] combine elements from random geometric graphs [14] and Chung-Lu graphs [10, 9]. Let $V = \{1, \dots, n\}$ be a set of vertices with positive weights w_1, \dots, w_n following a power law with exponent $\beta > 2$. Let W be their sum. Let \mathbb{T}^d be the d -dimensional torus for a fixed dimension $d \geq 1$ represented by the d -dimensional cube $[0, 1]^d$ where opposite boundaries are identified. For each vertex $v \in V$, let $x_v \in \mathbb{T}^d$ be a point drawn uniformly and independently at random. For $x, y \in \mathbb{T}^d$ let $\|x - y\|$ denote the L_∞ -norm on the torus, i.e. $\|x - y\| = \max_{1 \leq i \leq d} \min\{|x_i - y_i|, 1 - |x_i - y_i|\}$. Two vertices $u \neq v$ are independently connected with probability p_{uv} . For a positive temperature $0 < T < 1$,

$$p_{uv} = \min \left\{ 1, c \left(\frac{w_u w_v / W}{\|x_u - x_v\|^d} \right)^{1/T} \right\} \quad (1)$$

while for $T = 0$ a threshold variant of the model is obtained with

$$p_{uv} = \begin{cases} 1 & \text{if } \|x_u - x_v\| \leq c(w_u w_v / W)^{1/d}, \\ 0 & \text{else.} \end{cases}$$

The constant $c > 0$ controls the expected average degree. We note that the above formulation slightly deviates from the original definition; see Section 2.3 for more details.

2.2 Hyperbolic Random Graphs

HRGs [17] are generated by sampling random positions in the hyperbolic plane and connecting vertices that are close. More formally, let $V = \{1, \dots, n\}$ be a set of vertices. Let $\alpha > 1/2$ and $C \in \mathbb{R}$ be two constants, where α controls the power-law degree distribution with exponent $\beta = 2\alpha + 1 > 2$, and C determines the average degree \bar{d} . For each vertex $v \in V$, we sample a random point $p_v = (r_v, \theta_v)$ in the hyperbolic plane, using polar coordinates. Its angular coordinate θ_v is chosen uniformly from $[0, 2\pi)$ while its radius $0 \leq r_v < R$ with $R = 2 \log(n) + C$ is drawn according to the density function $f(r) = \frac{\alpha \sinh(\alpha r)}{\cosh(\alpha R) - 1}$. In the threshold case of HRGs two vertices $u \neq v$ are connected if and only if their distance is below R . The hyperbolic distance $d(p_u, p_v)$ is defined via $\cosh(d(p_u, p_v)) = \cosh(r_u) \cosh(r_v) - \sinh(r_u) \sinh(r_v) \cos(\theta_u - \theta_v)$.

The binomial variant adds a temperature $T \in [0, 1]$ to control the clustering, with lower temperatures leading to higher clustering. Two nodes $u, v \in V$ are then connected with probability $p_T(d(p_u, p_v))$ where $p_T(d) = (\exp[(d - R)/(2T)] + 1)^{-1}$. For $T \rightarrow 0$, the two definitions (threshold and binomial) coincide.

2.3 Comparison of GIRGs and HRGs

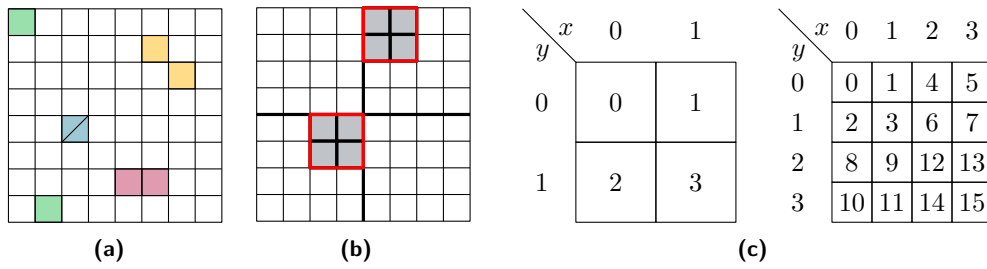
Bringmann et al. [7] show that the HRG model can be seen as a special case of the GIRG model in the following sense. Let d_{HRG} be the average degree of a HRG. Then there exist GIRGs with average degree d_{GIRG} and D_{GIRG} with $d_{\text{GIRG}} \leq d_{\text{HRG}} \leq D_{\text{GIRG}}$ such that they are sub- and supergraphs of the HRG, respectively. Moreover, d_{GIRG} and D_{GIRG} differ only by a constant factor. Formally, this is achieved by using the big- O notation instead of a single constant c for the connection probability. We call this the *generic GIRG framework*. It basically captures any specific model whose connection probabilities differ from Equation (1) by only a constant factor. From a theoretical point of view this is useful as proving something for the generic GIRG framework also proves it for any manifestation, including HRGs.

To see how HRGs fit into the generic GIRG framework, consider the following mapping [7]. Radii are mapped to weights $w_v = e^{(R-r_v)/2}$, and angles are scaled to fit on a 1-dimensional torus $x_v = \theta_v/(2\pi)$. One can then see that the hyperbolic connection probability $p_T(d)$ under the provided mapping deviates from Equation (1) by only a constant. Thus, c in Equation (1) can be chosen such that all GIRG probabilities are larger or smaller than the corresponding HRG probabilities, leading to the two average degrees d_{GIRG} and D_{GIRG} mentioned above. Bringmann et al. [7] note that the two constants, which they hide in the big- O notation, do not have to match. They leave it open if they match, converge asymptotically, or how large the interval between them is in practice. We investigate this empirically in Section 5.3.

3 Sampling Algorithm

As mentioned in the introduction, the core of our sampling algorithm is based on the algorithm by Bringmann et al. [7]. In the following, we first give a description of the core ideas and then work out the details that lead to an efficient implementation.

To explain the idea, we make two temporary assumptions and relax them in Section 3.1 and Section 3.2, respectively. For now, assume that all weights are equal and consider only the threshold variant $T = 0$. The task is to find all vertex pairs that form an edge, i.e., their distance is below the threshold $c(w_u w_v / W)^{1/d}$. Since all weights are equal, the threshold in this restricted scenario is the same for all vertex pairs. One approach to quickly identify adjacent vertices is to partition the ground space into a grid of cells. The size of the cells should be chosen, such that (I) the cells are as small as possible and (II) the diameter



■ **Figure 1** (a),(b) The grid used by weight bucket pairs with a connection probability threshold between 2^{-3} and 2^{-4} in two dimensions. (a) Each pair of colored cells represent neighbors. Note that the ground space is a torus and a cell is also a neighbor to itself. (b) The eight gray cells represent multiple distant cell pairs, which are replaced by one pair consisting of the red outlined parent cell pair. (c) Linearization of the cells on level 1 (left) and 2 (right) for $d = 2$.

of cells is larger than the threshold $c(w_u w_v / W)^{1/d}$. The latter implies that only vertices in neighboring cells can be connected thus narrowing down the search space. The former ensures that neighboring cells contain as few vertex pairs as possible reducing the number of comparisons. Figure 1a shows an example of such a grid for a 2-dimensional ground space.

3.1 Inhomogeneous Weights

Assume that we have vertices with two different weights w_1, w_2 , rather than one. As before, the cells should still be as small as possible while having a diameter larger than the connection threshold. However, there are three different thresholds now, one for each combination of weights. To resolve this, we can group the vertices by weight and use three differently sized grids to find the edges between them.

As GIRGs require not only two but many weights, considering one grid for every weight pair is infeasible. The solution is to discretize the weights by grouping ranges of weights into *weight buckets*. When searching for edges between vertices in two weight buckets, the pair of largest weights in these buckets provides the threshold for the cell diameter. This choice of the cell diameter satisfies property (II). Property (I) is violated only slightly, if the weight range within the bucket is not too large. Thus, each combination of two weight buckets uses a grid of cells, whose granularity is based on the maximum weight in the respective buckets.

As a tradeoff, we choose $\lceil \log_2 n \rceil$ many buckets which yields a sublinear number of grids. Moreover, the largest and smallest weight in a bucket are at most a factor two apart. Thus, the diameter of a cell is too large by at most a factor of four.

With this approach, a single vertex has to appear in grids of different granularity. To do this in an efficient manner, we recursively divide the space into ever smaller grid cells, leading to a hierarchical subdivision of the space. This hierarchy is naturally described by a tree. For a 2-dimensional ground space, each node has four children, which is why we call it *quadtree*. Note that each level of the quadtree represents a grid of different granularity. Moreover, the side length of a grid cell on level ℓ is $2^{-\ell}$. For a pair (i, j) of weight buckets, we then choose the level that fits best for the corresponding weights, i.e., the deepest level such that the diameter of each grid cell is above the connection threshold for the largest weights in bucket i and j , respectively. We call this level the *comparison level*, denoted by $CL(i, j)$. It suffices to insert vertices of a bucket into the deepest level among all its comparison levels. This level is called the *insertion level* and we denote it by $I(i)$. In Section 3.4, we discuss in detail how to efficiently access all vertices in a given grid cell belonging to a given weight bucket.

3.2 Binomial Variant of the Model

For $T > 0$, neighboring cell pairs are still easy to handle: a constant fraction of vertex pairs will have an edge and one can sample them by explicitly checking every pair. For distant cell pairs and a fixed pair of weight buckets, the distance between the cells yields an upper bound on the connection probability of included vertices; see Equation (1). The probability bound depends on both, the weight buckets and the cell pair distance, using the maximum weight within the buckets and the minimum distance between points in the cells. We note that the individual connection probabilities are only a constant factor smaller than the upper bound.

Knowing this, we can use geometric jumps to skip most vertex pairs [1]. The approach works as follows. Assume that we want to create an edge with probability \bar{p} for each vertex pair. For this process, we define the random variable X to be the number of vertex pairs we see until we add the next edge. Then X follows a geometric distribution. Thus, instead of throwing a coin for each vertex pair, we can do a single experiment that samples X from the geometric distribution and then skip X vertex pairs ahead. Since not all vertex pairs reach the upper bound \bar{p} , we accept encountered pairs with probability p_{uv}/\bar{p} to get correct results.

Although distant cell pairs are handled efficiently, their number is still quadratic, most of which yield no edges. To circumvent this problem, the sampling algorithm, yet again, uses a quadtree. In the quadratic set of cell pairs to compare for one weight bucket pair, non-neighboring cells are grouped together along the quadtree hierarchy. They are replaced by their parents as shown in Figure 1b until their parents become neighbors.

In conclusion, for each pair of weight buckets (i, j) the following two types of cell pairs have to be processed. Any two neighboring cell pairs on the comparison level $CL(i, j)$; and any distant cell pair with level larger or equal $CL(i, j)$ that has neighboring parents. The resulting set of distant and neighboring cell pairs for a fixed bucket pair partitions $\mathbb{T}^d \times \mathbb{T}^d$.

3.3 Efficiently Iterating Over Cell Pairs

The previous description sketches the algorithm as originally published. Here, we propose a refactoring that greatly simplifies the implementation and enables parallelization. We attribute a significant amount of HYPERGIRGS' speed up over EMBEDDER to this change.

Instead of first iterating over all bucket pairs and then over all corresponding cell pairs, we reverse this order. This removes the need to repeatedly determine the cell pairs to process for a given bucket pair. Instead it suffices to find the bucket pairs that process a given cell pair. This only depends on the level of the two cells and their type (neighboring or distant). Inverting the mapping from bucket pairs to cell pairs in the previous section yields the following. A neighboring cell pair on level ℓ is processed for bucket pairs with a comparison level of exactly ℓ . A distant cell pair on level ℓ (with neighboring parents) is processed for bucket pairs with a comparison level larger than or equal to ℓ . Thus, for each level of the quadtree we must enumerate all neighboring cell pairs, as well as distant cell pairs with neighboring parents. Algorithm 1 recursively enumerates exactly these cell pairs.

3.4 Efficient Access to Vertices by Bucket and Cell

A crucial part of the algorithm is to quickly access the set of vertices restricted to a weight bucket i and a cell A , which we denote by V_i^A . To this end, we linearize the cells of each level as illustrated in Figure 1c. This linearization is called Morton code [18] or z-order curve [20]. It has the nice properties that (I) for each cell in level ℓ , its descendants in level $\ell' > \ell$ in the quadtree appear consecutively; and (II) it is easy to convert between a cell's position in the linear order and its d -dimensional coordinates (see Section 4.2).

■ **Algorithm 1** Sample GIRG by Recursive Iteration of Cell Pairs.

Input: cell pair (A,B) ; initially called with A,B set to the root of the quadtree

```

1 forall bucket pairs  $(i, j)$  that process the cell pair  $(A, B)$  do
2   if  $A$  and  $B$  are neighbors then
3     emit each edge  $(u, v) \in V_i^A \times V_j^B$  with probability  $p_{uv}$ 
4   else
5     choose candidates  $S \subseteq V_i^A \times V_j^B$  using geometric jumps and  $\bar{p}$ 
6     emit each edge  $(u, v) \in S$  with probability  $p_{uv}/\bar{p}$ 
7 if  $A$  and  $B$  are neighbors and not maximum depth reached then
8   forall children  $X$  of  $A$  do
9     forall children  $Y$  of  $B$  do
10    recur( $X, Y$ )

```

We sort the vertices of a fixed weight bucket i by the Morton code of their containing cell on the insertion level $I(i)$, using arbitrary tie-breaking for vertices in the same cell. This has the effect that for any cell A with $\text{level}(A) \leq I(i)$, the vertices of V_i^A appear consecutive. Thus, to efficiently enumerate them, it suffices to know for each cell A the index of the first vertex in V_i^A . This can be precomputed using prefix sums leading to the following lemma.

► **Lemma 1.** *After linear preprocessing, for all cells A and weight buckets i with $\text{level}(A) \leq I(i)$, vertices in the set V_i^A can be enumerated in $\mathcal{O}(|V_i^A|)$.*

4 Implementation Details

The description in the previous section is an idealized version of the algorithm. For an actual implementation, there are some gaps to fill in. Omitting many minor tweaks, we want to sketch optimizations that are crucial to achieve a good practical run time in the following. More details on the sketched approaches can be found in the long version of this paper [3].

4.1 Estimating the Average Degree Parameter

Here, we sketch how to estimate the parameter c in Eq. (1) to achieve a given expected average degree. We estimate the constant based on the actual weights, not on their probability distribution. This leads to lower variance and allows user-defined weights.

We start with an arbitrary constant c , calculate the resulting expected average degree $\mathbb{E}[\bar{d}]$ and adjust c accordingly, using a modified binary search. This is possible, as $\mathbb{E}[\bar{d}]$ is monotone in c . We derive an exact formula for $\mathbb{E}[\bar{d}]$, depending on c and the weights. It cannot simply be solved for c , which is why we use binary search instead of a closed expression.

For the binary search, we need to evaluate $\mathbb{E}[\bar{d}]$ for different values of c . This is potentially problematic, as the formula for $\mathbb{E}[\bar{d}]$ sums over all vertex pairs. The issue preventing us from simplifying this formula is the minimum in the connection probability. Therefore, we first ignore the minimum and subtract an error term for those vertex pairs, where the minimum takes effect. The remaining hard part is to calculate this error term. Let E_S be the set of vertex pairs appearing in the error term and let S be the set of vertices with at least one partner in E_S . Although $|E_S|$ itself is sufficiently small, S is too large to determine E_S by iterating over all pairs in $S \times S$. We solve this by iterating over the vertices in S , sorted by weight. Then, for each vertex we encounter, the set of partners in E_S is a superset of the partners of the previous vertex with smaller weight, allowing us to compute $\mathbb{E}[\bar{d}]$ in $O(S)$.

4.2 Efficiently Encoding and Decoding Morton Codes

Recall from Section 3.4 that we linearize the d -dimensional grid of cells using Morton code. As vertex positions are given as d -dimensional coordinates, we have to convert the coordinates to Morton codes (i.e., the index in the linearization) and vice versa. This is done by bitwise interleaving the coordinates. For example, the 2-dimensional Morton code of the four-bit coordinates $a = a_3a_2a_1a_0$ and $b = b_3b_2b_1b_0$ is $a_3b_3a_2b_2a_1b_1a_0b_0$. We evaluated different encoding and decoding approaches via micro benchmarks. The fastest approach, at least on Intel processors, was an assembler instruction from BMI2 proposed by Intel in 2013 [16].

4.3 Generating HRGs Avoiding Expensive Mathematical Operations

The algorithm from Section 3 can be used to generate HRGs. The algorithm works conceptually the same, except that most formulas change. This has for example the effect that we no longer get a closed formula to determine the insertion level of a weight bucket or the comparison level of a bucket pair. Instead, one has to search them, by iterating over the levels of the quadtree. Furthermore, HRGs introduce many computationally expensive mathematical operations like the hyperbolic cosine. This can be mitigated as follows.

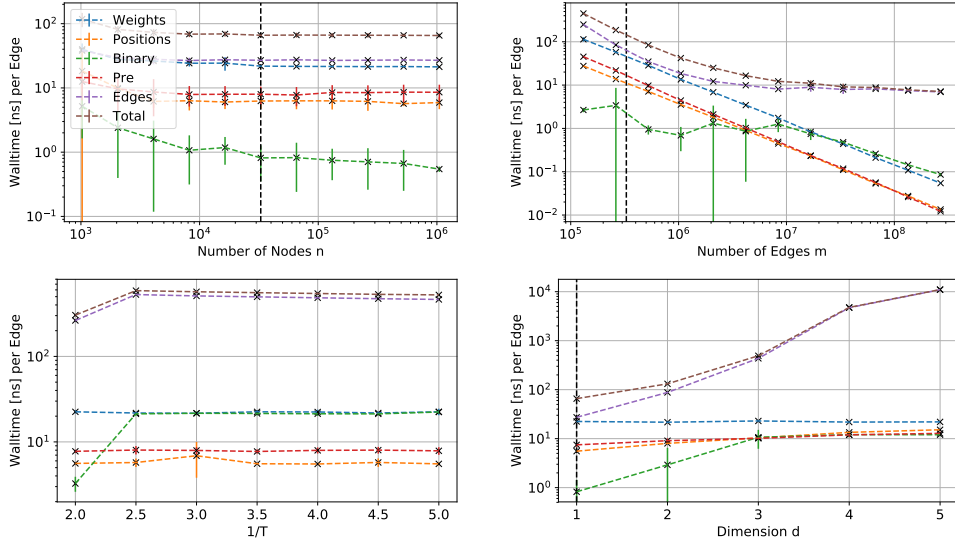
For the threshold model, an edge exists if the distance d is smaller than R . Considering how the hyperbolic distance is defined (Section 2.2), reformulating it to $\cosh(d) < \cosh(R)$ avoids the expensive $\operatorname{arccosh}$, while $\cosh(R)$ remains constant during execution and can thus be precomputed. Similar to recent threshold HRG generators, we compute intermediate values per vertex such that $\cosh(d)$ can be computed using only multiplication and addition [12, 21].

For the binomial model, evaluating the connection probability is a performance bottleneck. The straightforward way to sample edges is: compute the connection probability $p_T(d)$ depending on the distance, sample a uniform random value $u \in [0, 1]$, and create the edge if and only if $u < p_T(d)$. We can improve this by precomputing the inverse of $p_T(d)$ for equidistant values in $[0, 1]$. This lets us, for small ranges in $[0, 1]$, quickly access the corresponding range of distances. Changing the order, we first sample $u \in [0, 1]$, which falls in a range between two precomputed values, which in turn yields a range of distances. If the actual distance lies below that range, there has to be an edge and if it lies above, there is no edge. Only if it lies in the range, we actually have to compute the probability $p_T(d)$.

4.4 Parallelization

The algorithm has five steps: generate weights, generate positions, estimate the average degree constant, precompute the geometric data structure, and sample edges. The first two are trivial to parallelize. For estimating the constants, we parallelize the dominant computations with linear running time. To sample the edges, we make use of the fact that we iterate over cell pairs in a recursive manner. This can be parallelized by cutting the recursion tree at a certain level and distributing the loose ends among multiple processors.

For the preprocessing we have to do three subtasks: compute for each vertex its containing cells on its insertion level, sort the vertices according to their Morton code index, and compute the prefix sum for all cells. We parallelize all three tasks and optimize them by handling all weight buckets together, sorting by weight bucket first and Morton code second. This is done by encoding this criterion into integers that are sorted with parallel radix sort.



■ **Figure 2** Sequential run time for the steps of the GIRG sampling algorithm averaged over 10 iterations. Each plot varies a different model parameter deviating from a base configuration $d = 1$, $n = 2^{15}$, $T = 0$, $\beta = 2.5$, and $\bar{d} = 10$. The base configuration is indicated by a dashed vertical line.

5 Experimental Evaluation

We perform three types of experiments. In Section 5.1 we investigate the scaling behavior of our GIRG generator, broken down into the different tasks performed by the algorithm. In Section 5.2 we compare our HRG generator with existing generators. In Section 5.3 we experimentally investigate the difference between HRGs and their GIRG counterpart. Whenever a data point represents the mean over multiple iterations, our plots include error bars that indicate the standard deviation. Besides the implementation itself, all benchmarks and analysis scripts are also accessible in our source repository.

5.1 Scaling of the GIRG Generator

We investigate the scaling of the generator, broken down into five steps. 1. **(Weights)** Generate power-law weights. 2. **(Positions)** Generate points on \mathbb{T}^d . 3. **(Binary)** Estimate the constant controlling the average degree. 4. **(Pre)** Preprocess the geometric data structure (Section 3.4). 5. **(Edges)** Sample edges between all vertex pairs as described in Algorithm 1.

Figure 2 shows the sequential run time over the number of nodes n (top left), number of edges m (top right), temperature T (bottom right), and dimension d (bottom right). The performance is measured in nanoseconds per edge. Each data point represents the mean over 10 iterations. To make the measurements independent of the graph representation, we do not save the edges into RAM, but accumulate a checksum instead. Note that the top right plot increases the average degree, resulting in a decreased time per edge.

The empirical run times match the theoretical bounds: it is linear in n and m , grows exponentially in the dimension d , and is unaffected by the temperature T . The overall time is dominated by the edge sampling. Generating the weights includes expensive exponential functions, making it the slowest step after edge sampling. Generating the positions is significantly faster even for higher dimensions. For the parameter estimation using binary search, one can see that the run time never exceeds the time to generate the weights. For

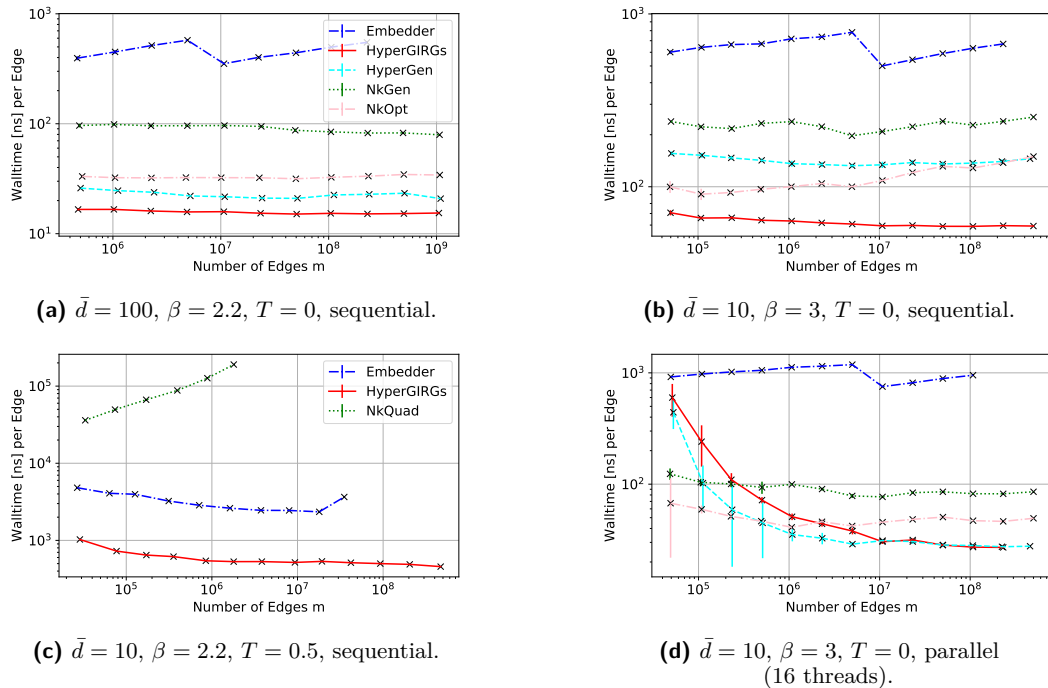


Figure 3 Comparison of HRG generators averaged over 5 iterations. (a), (b) Threshold variant for different average degrees \bar{d} and power-law exponents β . (c) Binomial variant with temperature $T = 0.5$. (d) The same configuration as (b) but utilizing multiple cores.

non-zero temperature T the performance of the binary search is similar to the generation of the weights, as it also requires exponential functions. The lower run times per edge for the increasing number of edges (top right) show that the run time is dominated by the number of nodes n . Only for very high average degrees, the cost per edge outgrows the cost per vertex.

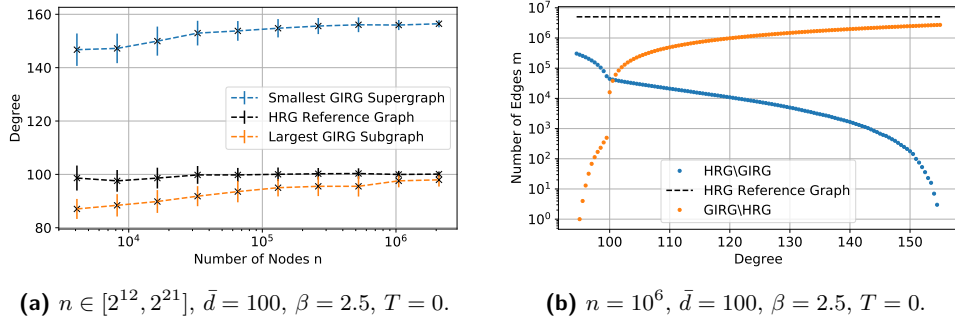
5.2 HRG Run Time Comparison

We evaluate the run time performance of HYPERGIRGs compared to the generators in Table 1, excluding the generators with high asymptotic run time as well as RHG and SRHG. RHG and SRHG are designed for distributed machines. Executed on a single compute node, the performance of the faster SRHG is comparable to HYPERGEN [12]. To avoid systematic biases between different graph representations, the implementations are modified² not to store the resulting graph. Instead, only the number of edges produced is counted and we ensure that the computation of incident nodes is not optimized away by the compiler.

We used different machines for our sequential and parallel experiments. The former are done on an *Intel Core i7-8700K* with 16 GB RAM, the latter on an *Intel Xeon CPU E5-2630 v3* with 8 cores (16 threads) and 64 GB RAM.

Our generator HYPERGIRGs is consistently faster than the competitors, independent of the parameter choices; see Figure 3a and 3b. Only for unrealistic average degrees (1k), HYPERGEN slightly outperforms HYPERGIRGs. Moreover, HYPERGIRGs beats EMBEDDER, the only other efficient generator supporting non-zero temperature, by an order of magnitude.

² The modifications are publicly available and referenced in our GitHub repository.



■ **Figure 4** Relation between the HRG and the GIRG model. (a) The values for d_{HRG} , d_{GIRG} , D_{GIRG} averaged over 50 iterations. (b) The number of missing ($\text{HRG} \setminus \text{GIRG}$) and additional ($\text{GIRG} \setminus \text{HRG}$) edges depending on the expected degree of the corresponding GIRG. It can be interpreted as a cross-section of one iteration in (a).

For higher temperatures, we compare our algorithm with the two other non-quadratic generators NKQUAD (included in NetworkKit) and EMBEDDER; see Figure 3c. We note that EMBEDDER uses a different estimation for R , which leads to an insignificant left-shift of the corresponding curve. In Figure 3c, one can clearly see the worse asymptotic running time of NKQUAD. Compared to EMBEDDER, HYPERGIRGS is consistently 4 times faster.

Figure 3d shows measurements for parallel experiments using 16 threads. The parameters coincide with Figure 3b. EMBEDDER does not support parallelization and is outperformed even more. For sufficiently large graphs, the fastest generator in this multi-core setting is HYPERGEN, which is specifically tailored towards parallel execution. Nonetheless, HYPERGIRGS shows comparable performance and overtakes the other two generators NKGGEN and NKOPT. We note that even on parallel machines, the sequential performance is of high importance: one often needs a large collection of graphs rather than a single huge instance. In this case, it is more efficient to run multiple instances of a sequential generator in parallel.

5.3 Difference Between HRGs and GIRGs

Recall from Section 2.3 that a HRG with average degree d_{HRG} has a corresponding GIRG sub- and supergraphs with average degrees d_{GIRG} and D_{GIRG} , respectively.

We experimentally determine, for given HRGs, the values for d_{GIRG} by decreasing the average degree of the corresponding GIRGs until it is a subgraph of the HRG. Analogously, we determine the value for D_{GIRG} . We focus on the threshold variant of the models, as this makes the coupling between HRGs and GIRGs much simpler (the graph is uniquely determined by the coordinates). Figure 4a shows d_{GIRG} and D_{GIRG} , compared to d_{HRG} for growing n . One can see that d_{GIRG} and D_{GIRG} are actually quite far apart. They in particular do not converge to the same value for growing n . However, at least d_{GIRG} seems to approach d_{HRG} . This indicates that every HRG corresponds to a GIRG subgraph that is missing only a sublinear fraction of edges. On the other hand, the average degree of the GIRG has to be increased by a lot to actually contain all edges also contained in the HRG.

Figure 4b gives a more detailed view for a single HRG. Depending on the average degree of the GIRG, it shows how many edges the GIRG lacks and how many edges the GIRG has in addition to the HRG. For degree 100, the GIRG contains about 38k additional and lacks about 42k edges. These are rather small numbers compared to the 50 M edges of the graphs.

References

- 1 Joachim H. Ahrens and Ulrich Dieter. Sequential Random Sampling. *ACM Transactions on Mathematical Software*, 11(2):157–169, 1985. doi:10.1145/214392.214402.
- 2 Rodrigo Aldecoa, Chiara Orsini, and Dmitri Krioukov. Hyperbolic Graph Generator. *Computer Physics Communications*, 196:492–496, 2015. doi:10.1016/j.cpc.2015.05.028.
- 3 Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, Ulrich Meyer, Manuel Penschuck, and Christopher Weyand. Efficiently Generating Geometric Inhomogeneous and Hyperbolic Random Graphs. *CoRR*, abs/1905.06706, 2019. arXiv:1905.06706.
- 4 Thomas Bläsius, Cedric Freiberger, Tobias Friedrich, Maximilian Katzmann, Felix Montenegro-Retana, and Marianne Thieffry. Efficient Shortest Paths in Scale-Free Networks with Underlying Hyperbolic Geometry. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107, pages 20:1–20:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.20.
- 5 Thomas Bläsius, Tobias Friedrich, Anton Krohmer, and Sören Laue. Efficient Embedding of Scale-Free Graphs in the Hyperbolic Plane. *IEEE/ACM Transactions on Networking*, 26(2):920–933, 2018. doi:10.1109/TNET.2018.2810186.
- 6 OpenMP Architecture Review Board. OpenMP application program interface version 5.0, 2018. URL: <https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf>.
- 7 Karl Bringmann, Ralph Keusch, and Johannes Lengler. Geometric Inhomogeneous Random Graphs. *Theoretical Computer Science*, 760:35–54, 2019. doi:10.1016/j.tcs.2018.08.014.
- 8 Deepayan Chakrabarti and Christos Faloutsos. Graph Mining: Laws, Generators, and Algorithms. *ACM Comput. Surv.*, 38(1), 2006. doi:10.1145/1132952.1132954.
- 9 Fan Chung and Linyuan Lu. Connected Components in Random Graphs with Given Expected Degree Sequences. *Annals of Combinatorics*, 6(2):125–145, 2002. doi:10.1007/PL00012580.
- 10 Fan Chung and Linyuan Lu. The Average Distances in Random Graphs with Given Expected Degrees. *Proceedings of the National Academy of Sciences*, 99(25):15879–15882, 2002. doi:10.1073/pnas.252631999.
- 11 Tobias Friedrich and Anton Krohmer. On the Diameter of Hyperbolic Random Graphs. *SIAM Journal on Discrete Mathematics*, 32(2):1314–1334, 2018. doi:10.1137/17M1123961.
- 12 Daniel Funke, Sebastian Lamm, Ulrich Meyer, Manuel Penschuck, Peter Sanders, Christian Schulz, Darren Strash, and Moritz von Looz. Communication-free massively distributed graph generation. *Journal of Parallel and Distributed Computing*, 131:200–217, 2019. doi:10.1016/j.jpdc.2019.03.011.
- 13 Daniel Funke, Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Moritz von Looz. Communication-Free Massively Distributed Graph Generation. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 336–347, 2018. doi:10.1109/IPDPS.2018.00043.
- 14 Edgar N. Gilbert. Random Plane Networks. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):533–543, 1961. doi:10.1137/0109045.
- 15 Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. Random Hyperbolic Graphs: Degree Sequence and Clustering. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 573–585, 2012. doi:10.1007/978-3-642-31585-5_51.
- 16 Intel Corporation. *Intel 64 and IA-32 Architectures Developer’s Manual*, 2019.
- 17 Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic Geometry of Complex Networks. *Physical Review E*, 82:036106, 2010. doi:10.1103/PhysRevE.82.036106.
- 18 Guy M Morton. A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing. Technical report, International Business Machines Company New York, 1966. URL: <https://domino.research.ibm.com/library/cyberdig.nsf/0/0dabf9473b9c86d48525779800566a3970openDocument>.
- 19 Tobias Müller and Merlijn Staps. The Diameter of KPKVB Random Graphs. *CoRR*, abs/1707.09555, 2017. arXiv:1707.09555.

- 20 J. A. Orenstein and T. H. Merrett. A Class of Data Structures for Associative Searching. In *ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS)*, pages 181–190, 1984. doi:10.1145/588011.588037.
- 21 Manuel Penschuck. Generating Practical Random Hyperbolic Graphs in Near-Linear Time and with Sub-Linear Memory. In *International Symposium on Experimental Algorithms (SEA)*, volume 75, pages 26:1–26:21, 2017. doi:10.4230/LIPIcs.SEA.2017.26.
- 22 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.
- 23 Moritz von Looz and Henning Meyerhenke. Querying Probabilistic Neighborhoods in Spatial Data Sets Efficiently. In *International Workshop on Combinatorial Algorithms (IWOCA)*, pages 449–460, 2016. doi:10.1007/978-3-319-44543-4_35.
- 24 Moritz von Looz, Henning Meyerhenke, and Roman Prutkin. Generating Random Hyperbolic Graphs in Subquadratic Time. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 467–478, 2015. doi:10.1007/978-3-662-48971-0_40.
- 25 Moritz von Looz, Mustafa Safa Özdayi, Sören Laue, and Henning Meyerhenke. Generating Massive Complex Networks with Hyperbolic Geometry Faster in Practice. In *IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, 2016. doi:10.1109/HPEC.2016.7761644.
- 26 Duncan J. Watts and Steven H. Strogatz. Collective Dynamics of “Small-World” Networks. *Nature*, 393:440–442, 1998. doi:10.1038/30918.

Randomized Incremental Construction of Delaunay Triangulations of Nice Point Sets

Jean-Daniel Boissonnat

INRIA Sophia-Antipolis, Université Côte d’Azur, Nice, France
jean-daniel.boissonnat@inria.fr

Olivier Devillers 

INRIA, CNRS, Loria, Université de Lorraine, Nancy, France
olivier.devillers@inria.fr

Kunal Dutta 

INRIA Sophia-Antipolis, Université Côte d’Azur, Nice, France
kunal.dutta@inria.fr

Marc Glisse

INRIA, Université Paris-Saclay, France
marc.glisse@inria.fr

Abstract

Randomized incremental construction (RIC) is one of the most important paradigms for building geometric data structures. Clarkson and Shor developed a general theory that led to numerous algorithms that are both simple and efficient in theory and in practice.

Randomized incremental constructions are most of the time space and time optimal in the worst-case, as exemplified by the construction of convex hulls, Delaunay triangulations and arrangements of line segments. However, the worst-case scenario occurs rarely in practice and we would like to understand how RIC behaves when the input is nice in the sense that the associated output is significantly smaller than in the worst-case. For example, it is known that the Delaunay triangulations of nicely distributed points on polyhedral surfaces in E^3 has linear complexity, as opposed to a worst-case quadratic complexity. The standard analysis does not provide accurate bounds on the complexity of such cases and we aim at establishing such bounds in this paper. More precisely, we will show that, in the case of nicely distributed points on polyhedral surfaces, the complexity of the usual RIC is $O(n \log n)$, which is optimal. In other words, without any modification, RIC nicely adapts to good cases of practical value.

Our proofs also work for some other notions of nicely distributed point sets, such as (ε, κ) -samples. Along the way, we prove a probabilistic lemma for sampling without replacement, which may be of independent interest.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases Randomized incremental construction, Delaunay triangulations, Voronoi diagrams, polyhedral surfaces, probabilistic analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.22

Related Version <https://hal.inria.fr/hal-01950119>

Funding This work has been partially supported by the European Research Council under Advanced Grant 339025 GUDHI (Algorithmic Foundations of Geometric Understanding in Higher Dimensions) and by grant ANR-17-CE40-0017 of the French National Research Agency (ANR project ASPAG).

Acknowledgements The authors would like to acknowledge the referees for their helpful comments which helped to improve the presentation of the paper, in particular a simpler proof of Lemma 9.



© Jean-Daniel Boissonnat, Olivier Devillers, Kunal Dutta, and Marc Glisse;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 22; pp. 22:1–22:13

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The *randomized incremental construction* (RIC) is an algorithmic paradigm introduced by Clarkson and Shor [10], which has since found immense applicability in computational geometry, e.g., [21, 20]. The general idea is to process the input points sequentially in a random order, and to analyze the expected complexity of the resulting procedure. The theory developed by Clarkson and Shor is quite general and has led to numerous algorithms that are simple and efficient, both in theory and in practice. On the theory side, randomized incremental constructions are most of the time space and time optimal in the worst-case, as exemplified by the construction of convex hulls, Delaunay triangulations and arrangements of line segments. Randomized incremental constructions appear also to be very efficient in practice, which, together with their simplicity, make them the most popular candidates for implementations. Not surprisingly, the CGAL library includes several randomized incremental algorithms, e.g., for computing Delaunay triangulations [22].

This paper aims at extending the analysis of RIC to the case of *nice-case* complexity. More precisely, our goal is to understand how randomized incremental constructions behave when the input is nice in the sense that the associated construction is significantly smaller than in the worst-case.

In this paper, we shall consider the case where the underlying space is a polyhedral surface in \mathbb{E}^3 . This is a commonly-occurring practical scenario in e.g., surface reconstruction [1, 6], and has been studied by several authors [2, 4, 5, 17]. Further, we need a model of good point sets to describe the input data and analyze the algorithm. This will be done through the notion of ε -nets. When we enforce such a hypothesis of “nice” distribution of the points in space, a result of Attali and Boissonnat [4] ensures that the complexity of the Delaunay triangulation is linear in the number of points. Unfortunately, to be able to control the complexity of the usual randomized incremental algorithms [3, 9, 10, 12], it is not enough to control the final complexity of the Delaunay triangulation. We need to control also the complexity of the triangulation of random subsets. One might expect that a random subsample of size k of an ε -net is also an ε' -net for $\varepsilon' = \varepsilon\sqrt{\frac{n}{k}}$. Actually this is not quite true, it may happen with reasonable probability that a ball of radius $O(\varepsilon')$ contains $\Omega(\log k / \log \log k)$ points or that a ball of radius $\Omega(\varepsilon'\sqrt{\log k})$ does not contain any point. However, it can only be shown that such a subsample is an $(\frac{\varepsilon'}{\log(1/\varepsilon)})$ -covering and an $(\varepsilon' \log(1/\varepsilon))$ -packing, with high probability. Thus this approach can transfer the complexity of an ε -net to the one of a random subsample of an ε -net but with an extra multiplicative factor of $\Omega(\log 1/\varepsilon) = \Omega(\log n)$. It follows that, in the case we consider, the standard analysis does not provide accurate bounds on the complexity of the (standard) randomized incremental construction. Our results are based on proving that the above bad scenarios occur rarely, and the algorithm achieves optimal run-time complexity, in expectation.

Related Work

The Delaunay triangulations of nicely-distributed points have been studied since the 50's, e.g., by Meijering [18], and later by Møller [19], Dwyer [13, 14], and others. Erickson [15, 16] proved upper and lower bounds for point samples with bounded *spread* (the ratio between the maximum to minimum distance between any two points) in \mathbb{E}^3 . For polyhedral surfaces, Golin and Na [17] gave an $O(n \log^4 n)$ bound for Poisson-distributed points. Attali and Boissonnat [4] showed that for (ε, κ) -samples, the complexity of the Delaunay triangulation is linear. Under some extra assumptions, this was extended by Amenta, Attali, and Devillers [2] to higher-dimensional polyhedral surfaces. Attali, Boissonnat, and Lieutier [5] proved an $O(n \log n)$ bound for (ε, κ) -samples on *smooth* surfaces in \mathbb{E}^3 .

Except for a few authors such as Dwyer [14] and Erickson [16], most of the above results discuss only the combinatorial aspects and not the algorithmic ones. For Poisson and uniformly distributed point samples, we observe that the standard analysis of the RIC procedure immediately implies an optimal bound on the expected run-time. However, for deterministic notions of nice distributions such as ε -nets, (ε, κ) -samples, and bounded spread point sets, the standard RIC analysis is not optimal, since, as we observed, it gives at least an extra logarithmic factor for (ε, κ) -samples and even worse for bounded spread point-sets, as stated in an open problem by Erickson [16].

Our Contribution

For ε -nets on polyhedral surfaces in \mathbb{E}^3 , we establish tight bounds on the complexity of random subsamples of any given size. Using this, we show that the complexity of the usual RIC is $O(n \log n)$, which is optimal. Hence, without any modification, the standard RIC nicely adapts to polyhedral surfaces in \mathbb{E}^3 .

Our technical developments rely on a general bound for the probability of certain non-monotone events in sampling without replacement, which may be of independent interest. We use this together with a geometric construction that, given a point p on a plane P , and a threshold radius r , allows us to bound the probability of existence of any empty disk in P with radius at least r , having p on its boundary. Lastly, the boundary effects need to be explicitly controlled, which requires a careful handling along the lines of the result of Attali and Boissonnat [4], along with some new ideas which we develop. (For a more detailed outline of the ideas, see the discussion in Section 3).

We remark that though we focus on polyhedral surfaces in \mathbb{E}^3 in this paper, our techniques are more general, and can be extended to e.g., ε -nets on d -dimensional flat torii, etc., which we do in the full version of this paper.

Outline

The rest of the paper is as follows. In Section 2, we define the basic concepts of Delaunay triangulation, ε -nets and random samples. We state our theorems and their proofs in Section 3. In Section 4, we give the proofs of some technical lemmas needed for the proofs of our theorems. Proofs missing from the main sections can be found in the full version of this paper [7].

2 Background

2.1 Notations

We shall use $\|\cdot\|$ to denote the Euclidean ℓ_2 norm. We denote by $\Sigma(p, r)$, $B(p, r)$ and $B[p, r]$, the sphere, the open ball, and the closed ball of center p and radius r respectively. For $x \in \mathbb{E}^2$, $y \geq 0$, $D(x, r)$ denotes the disk with center x and radius r , i.e. the set of points $\{y \in \mathbb{E}^2 : \|y - x\| < r\}$, and similarly $D[x, r]$ denotes the corresponding closed disk.

For an event \mathcal{E} in some probability space Ω , we use $\mathbb{1}_{[\mathcal{E}]}$ to denote the indicator variable $\mathbb{1}_{[\mathcal{E}]} = \mathbb{1}_{[\mathcal{E}]}(\omega)$ which is 1 whenever $\omega \in \mathcal{E}$, and zero otherwise. We use $[n]$ to mean the set $\{1, 2, \dots, n\}$. Given a discrete set A , $\#(A)$ denotes its cardinality and, for $k \in \mathbb{Z}^+$, $\binom{A}{k}$ denotes the set of k -sized subsets of A . Given an event A in some probability space, $\mathbb{P}[A]$ denotes the probability of A occurring. For a random variable Z in a probability space, $\mathbb{E}[Z]$ denotes the expected value of Z . Lastly, $e = 2.7182\dots$ denotes the base of the natural logarithm.

2.2 ε -nets

A set \mathcal{X} of n points in a metric space \mathcal{M} , is an ε -packing if any pair of points in \mathcal{X} are at least distance ε apart, and an ε -cover if each point in \mathcal{M} is at distance at most ε from some point of \mathcal{X} . \mathcal{X} is an ε -net if it is an ε -cover and an ε -packing simultaneously.

The definition of an ε -net applies for any metric space. In the case of the Euclidean metric, we can prove some additional properties, which will be given in Section 3.

2.3 Delaunay Triangulation

For simplicity of exposition and no real loss of generality, all finite point sets considered in this paper will be assumed to be in *general position*, i.e. there are no 5 points lying on a sphere in \mathbb{E}^3 , and no plane has a set of 4 points lying on a circle. Given a set \mathcal{X} in some ambient topological space, the *Delaunay complex* of \mathcal{X} is the (abstract) simplicial complex with vertex set \mathcal{X} which is the nerve of the Voronoi diagram of \mathcal{X} , that is, a simplex σ (of arbitrary dimension) belongs to $Del(\mathcal{X})$ iff the Voronoi cells of its vertices have a non empty common intersection. Equivalently, σ can be circumscribed by an empty ball, i.e. a ball whose bounding sphere contains the vertices of σ and whose interior contains no points of \mathcal{X} .

For point sets in \mathbb{E}^3 in general position, the Delaunay complex embeds in \mathbb{E}^3 and is a triangulation of the space.

2.4 Polyhedral Surfaces in \mathbb{E}^3

A *polyhedral surface* \mathcal{S} in \mathbb{E}^3 is a collection of a finite number of polygons $F \subset \mathcal{S}$, called *facets*, which are pairwise disjoint or meet along an edge. In this paper, \mathcal{S} will denote an arbitrary but fixed polyhedral surface, with C facets, and having total length of the boundaries of its faces L and total area of its faces A . Any non-convex polygonal facet $F \in \mathcal{S}$ can be triangulated and replaced in \mathcal{S} by the collection of triangular facets obtained. This will only change the total length L of the boundaries, which, for a given triangulation, still depends only on the original surface \mathcal{S} . Thus without any real loss of generality, we can (and shall) assume the facets of \mathcal{S} are convex.

2.5 Randomized Incremental Construction and Random Subsamples

For the algorithmic complexity aspects, we state a version of a standard theorem for the RIC procedure, (see e.g., [11]). We first need a necessary condition for the theorem. When a new point p is added to an existing triangulation, a *conflict* is defined to be a previously existing simplex whose circumball contains p .

► **Condition 1.** *At each step of the RIC, the set of simplices in conflict can be removed and the set of newly introduced conflicts can be computed in time proportional to the number of conflicts.*

We now come to the general theorem on the algorithmic complexity of RIC using the Clarkson-Shor technique (see e.g., Devillers [11] Theorem 5(1,2)).

► **Theorem 2.** *Let $F(s)$ denote the expected number of simplices that appear in the Delaunay triangulation of a uniform random sample of size s , from a given point set P . Then, if Condition 1 holds and $F(s) = O(s)$, we have*

- (i) *The expected space complexity of computing the Delaunay triangulation is $O(n)$.*
- (ii) *The expected time complexity of computing the Delaunay triangulation is $O(n \log n)$.*

A subset \mathcal{Y} of set \mathcal{X} is a *uniform random sample* of \mathcal{X} of size s if \mathcal{Y} is any possible subset of \mathcal{X} of size s with equal probability.

In order to work with uniform random samples, we shall prove a lemma on the uniformly random sampling distribution or *sampling without replacement*, which is stated below, and will be a key probabilistic component of our proofs. The lemma provides a bound on the probability of a non-monotone compound event, that is, if the event holds true for a fixed set of k points, there could exist supersets as well as subsets of the chosen set for which the event does not hold. This may well be of general interest, as most natural contiguity results with Bernoulli (i.e. independent) sampling, are for monotone events.

► **Lemma 3.** *Given $a, b, c \in \mathbb{Z}^+$, with $2b \leq a \leq c$, $t \leq c$. Let C be a set, and B and T two disjoint subsets of C . If A is a random subset of C , chosen uniformly from all subsets of C having size a , the probability that A contains B and is disjoint from T , is at most $(\frac{a}{c})^b \left(1 - \frac{t}{c-b}\right)^{a-b} \leq (\frac{a}{c})^b \cdot \exp\left(-\frac{at}{2c}\right)$, where a, b, c are the cardinalities of A, B , and C respectively, and the cardinality of T is at least t .*

Proof. The total number of ways of choosing the random sample A is $\binom{c}{a}$. The number of ways of choosing A such that $B \subset A$ and $T \cap A = \emptyset$, is $\binom{c-b-t}{a-b}$. Therefore the required probability is

$$\begin{aligned} \mathbb{P}[B \subset A, T \cap A = \emptyset] &= \frac{\binom{c-b-t}{a-b}}{\binom{c}{a}} \\ &= \frac{\prod_{i=0}^{b-1} (a-i) \prod_{i=b}^{a-1} (a-i)}{\prod_{i=0}^{b-1} (c-i) \prod_{i=b}^{a-1} (c-i)} \cdot \frac{\prod_{i=0}^{a-b-1} (c-b-t-i)}{\prod_{i=0}^{a-b-1} (a-b-i)} \\ &= \frac{\prod_{i=0}^{b-1} (a-i) \prod_{i=0}^{a-b-1} (c-b-t-i)}{\prod_{i=0}^{b-1} (c-i) \prod_{i=0}^{a-b-1} (c-b-i)} \\ &= (a/c)^b \frac{\prod_{i=0}^{b-1} (1-i/a)}{\prod_{i=0}^{b-1} (1-i/c)} \left(1 - \frac{t}{c-b}\right)^{a-b} \left(\frac{\prod_{i=0}^{a-b-1} (1 - \frac{i}{c-b-t})}{\prod_{i=0}^{a-b-1} (1 - \frac{i}{c-b})}\right) \\ &\leq (a/c)^b \left(1 - \frac{t}{c-b}\right)^{a-b}, \end{aligned}$$

where in the last step, observe that for the product $\frac{\prod_{i=0}^{b-1} (1-i/a)}{\prod_{i=0}^{b-1} (1-i/c)}$ for each i , the term $(1-i/a)$ in the numerator is smaller than the corresponding term $(1-i/c)$ in the denominator, since $a \leq c$. A similar observation holds for the product $\left(\frac{\prod_{i=0}^{a-b-1} (1 - \frac{i}{c-b-t})}{\prod_{i=0}^{a-b-1} (1 - \frac{i}{c-b})}\right)$.

Now, observe that $\left(1 - \frac{t}{c-b}\right)^{a-b} \leq \exp\left(-\left(\frac{t(a-b)}{c-b}\right)\right) \leq \exp\left(-\left(\frac{at}{2c}\right)\right)$, if $b \leq a/2$ and $b < c$. ◀

3 Results and Main Proofs

We show that the expected complexity of the Delaunay triangulation of a uniformly random subsample of an ε -net on a polyhedral surface is linear in the size of the subsample:

► **Theorem 4.** *Let $\varepsilon \in (0, 1]$, \mathcal{X} be an ε -net on a polyhedral surface \mathcal{S} , having n points and let $\mathcal{Y} \subset \mathcal{X}$ be a random sub-sample of \mathcal{X} having size s . Then, in expectation, the Delaunay triangulation $Del(\mathcal{Y})$ of \mathcal{Y} on \mathcal{S} has $O(s)$ simplices.*

Algorithmic Bounds

We next use the above combinatorial bound to get the space and time complexity of the randomized incremental construction of the Delaunay triangulation of an ε -net on a polyhedral surface in \mathbb{E}^3 .

► **Theorem 5** (Randomized incremental construction). *Let $\varepsilon \in (0, 1]$, and let \mathcal{X} be an ε -net in general position over a fixed polyhedral surface $\mathcal{S} \subset \mathbb{E}^3$, then the randomized incremental construction of the Delaunay triangulation takes $O(n \log n)$ expected time and $O(n)$ expected space, where $n = \#\mathcal{X}$ and the constant in the big O depends on (and only on) \mathcal{S} .*

► **Remark 6.** Theorem 4 also works for the case when the random sample is a Bernoulli sample of parameter $q := \frac{s}{n}$.

► **Remark 7.** Our results can be extended to other types of good samples, e.g., the weaker notion of (ε, κ) -samples for which any ball of radius ε contains at least one point and at most κ points. If we fix $\kappa = \kappa_0 = 2^{O(d)}$, we get exactly the same result. The bounds can be straightforwardly adapted to accommodate other values of κ .

Before presenting the proof of Theorem 4, we briefly discuss the outline of the proof.

Main Ideas

Our overall strategy will be to mesh the proof of Attali and Boissonnat [4] with some new ideas which are needed for random subsamples of ε -nets. Briefly, Attali and Boissonnat reduce the problem to counting the Delaunay edges of the point sample, which they do by distinguishing between *boundary* points, which lie in a strip of width ε near the boundaries of the facets of the polyhedral surface, and the other points, called *interior* points. For boundary points, they allow all possible edges. For interior points, the case of edges with endpoints on the same facet is easy to handle, while geometric constructions are required to handle the case of endpoints on different facets, or that of edges with one endpoint in the interior and another on the boundary.

However, we shall need to introduce a couple of new ideas. Firstly, an edge can have multiple balls passing through its endpoints and, as soon as one of these balls is empty, the edge is in the triangulation. To bound therefore, the probability of a potential edge appearing in the triangulation, we need to simultaneously bound the probability of any of these balls being empty. To ensure this, we use a geometric construction (see Lemma 17). Basically, the idea is to build a constant-sized packing of a sphere centered on a given point, using large balls, such that any sphere of a sufficiently large radius which passes through the point, must contain a ball from the packing.

Secondly, since we have randomly spaced points at the boundaries, boundary effects are no longer necessarily contained in the fixed strip of width ε around the boundary, and could potentially penetrate deep into the interior. To handle this, we generalize the fixed-width strip using the notion of *levels* of a facet. We then use a probabilistic, rather than deterministic, classification of boundary and interior points. The new classification is based on the level of a point and the radius of the largest empty disk passing through it.

Recall the definitions of \mathcal{X} , \mathcal{Y} and \mathcal{S} from Theorem 4. We shall use κ to denote the maximum number of points of a given point set in a disk of radius 2ε . When \mathcal{X} is an ε -net, κ is at most 25, using a packing argument (the maximum number of disjoint discs of radius $\varepsilon/2$ that can be packed in a disc of radius $2\varepsilon + \varepsilon/2$, is $\frac{\pi(5\varepsilon/2)^2}{\pi(\varepsilon/2)^2} \leq 25$). We define $q := \frac{s}{n}$, and $\delta := \varepsilon/\sqrt{q}$. For a curve Γ , $l(\Gamma)$ denotes its length. For a subset of a surface $R \subset \mathcal{S}$, $a(R)$ denotes the area of R . For sets $A, B \subset \mathbb{E}^3$, $A \oplus B$ denotes the Minkowski sum of A and B , i.e. the set $\{x + y : x \in A, y \in B\}$. For convenience, the special case $A \oplus B(0, r)$ shall be denoted by $A \oplus r$.

We next present some general lemmas, which will be needed in the proofs of the main lemmas.

Level Sets, Boundary Points and Interior Points

We now introduce some definitions which will play a central role in the analysis. First we define the notion of levels. Given facet $F \in \mathcal{S}$ and $k \geq 0$, define the *level set* $L_{\leq k} := F \cap (\partial F \oplus 2^k \delta)$. $L_{=k} := L_{\leq k} \setminus L_{\leq k-1}$. For $x \in \mathcal{X}$, the *level* of x , denoted $Lev(x)$, is k such that $x \in L_{=k}$. Let $L_{\leq k}(\mathcal{X}), L_{=k}(\mathcal{X})$ denote $L_{\leq k} \cap \mathcal{X}, L_{=k} \cap \mathcal{X}$ respectively. Note that for $x \in L_{=k}, k \geq 1$, the distance $d(x, \partial F) \in (2^{k-1}\delta, 2^k\delta]$. Hence, if $Lev(x) = k, D(x, 2^{k-1}\delta) \subset F$. For $k = 0, d(x, \partial F) \in [0, \delta]$.

Given $x \in F$ having $Lev(x) = k, x$ is a *boundary point* or $x \in Bd_F(\mathcal{Y})$ if $k = 0$ or if there exists an empty disk (w.r.t. \mathcal{Y}) of radius greater than $2^{k-1}\delta$, whose boundary passes through x . x is an *interior point* or $x \in Int_F(\mathcal{Y})$ if and only if $x \in \mathcal{Y} \setminus Bd_F(\mathcal{Y})$. In general, $x \in Bd_{\mathcal{S}}(\mathcal{Y})$ if $x \in Bd_F(\mathcal{Y})$ for some $F \in \mathcal{S}$, and $x \in Int_{\mathcal{S}}(\mathcal{Y})$ is defined similarly.

The above bi-partition induces a classification of potential edges of $Del(\mathcal{Y})$, depending on whether the end-points are boundary or interior points. Let E_1 denote the set of edges whose end points are two boundary points. Let E_2 denote the set of edges having as end-points, two interior points of the same facet of \mathcal{S} . Let E_3 denote the set of edges having as end-points, two interior points of different facets of \mathcal{S} . Let E_4 denote the set of edges having an interior point and a boundary point as end-points.

We have the following lemmas, to be proved in section 4.2.

► **Lemma 8.** $\mathbb{E}[\#(E_1)] \leq O(1) \cdot (\kappa^2 L^2 / A) \cdot s$.

► **Lemma 9.** $\mathbb{E}[\#(E_2)] \leq O(1) \cdot \kappa s$.

► **Lemma 10.** $\mathbb{E}[\#(E_3)] \leq O(1) \cdot (C - 1) \cdot \kappa s$.

► **Lemma 11.** $\mathbb{E}[\#(E_4)] \leq O(1) \cdot \frac{\kappa^2 L^2}{A} s$.

Given the above lemmas, the proof of Theorem 4 follows easily.

Proof of Theorem 4. As in [4] (Section 4), by Euler's formula, the number of tetrahedra $t(Del(\mathcal{Y}))$ in the Delaunay triangulation of \mathcal{S} , is at most $e(Del(\mathcal{Y})) - \#(\mathcal{Y}) = e(Del(\mathcal{Y})) - s$, where $e(Del(\mathcal{Y}))$ is the number of edges in the Delaunay triangulation. Therefore, it suffices to count the edges of $Del(\mathcal{Y})$. Next, observe that any point $x \in \mathcal{Y}$ is either a boundary or an interior point, that is $Bd_{\mathcal{S}}(\mathcal{Y}) \sqcup Int_{\mathcal{S}}(\mathcal{Y}) = \mathcal{Y}$. An edge in $Del(\mathcal{Y})$, therefore, can be either between two points in $Bd_{\mathcal{S}}(\mathcal{Y})$, or two points in $Int_{\mathcal{S}}(\mathcal{Y})$, or between a point in $Bd_{\mathcal{S}}(\mathcal{Y})$ and another in $Int_{\mathcal{S}}(\mathcal{Y})$. The case of a pair of points in $Int_{\mathcal{S}}(\mathcal{Y})$ is further split based on whether the points belong to the same facet of \mathcal{S} or different facets. Thus using the above exhaustive case analysis, the proof follows simply by summing the bounds of Lemmas 8 to 11. ◀

Next, we show how Theorem 4 implies bounds on the computational complexity of constructing Delaunay triangulations of ε -nets. Our main tool shall be Theorem 2. However, we need to show first that Condition 1 holds. The standard proof of this (see e.g., [10], [9], also the discussion in [8](Section 2.2 D)) is sketched below.

Now we come to the proof of Theorem 5.

Proof of Theorem 5. To verify that Condition 1 indeed holds in the case of polyhedral surfaces, observe first that the union \mathcal{C}_p of the simplices in conflict with a new point p is a connected set. Therefore, walking on the adjacency graph of the simplices by rotating around the edge or triangle shared between two adjacent faces on the boundary of \mathcal{C}_p , is enough to yield the set of new conflicts. Now Theorem 2 can be applied to get the claimed result. ◀

4 Proofs of Lemmas 8–11

Before proving Lemmas 8–11, we need a few technical lemmas.

4.1 Some Technical Lemmas

The following geometric and probabilistic lemmas prove certain properties of ε -nets on polyhedral surfaces and random subsets, as well as exploit the notion of boundary and interior points to get an exponential decay for boundary effects penetrating into the interior.

► **Lemma 12.** *Given $a > 0$, $b \in (0, 1)$, the sum $\sum_{n \in \mathbb{Z}_+} 2^{an} \cdot \exp(-b \cdot 2^{an})$ is at most $\frac{2 \log_2(1/b)}{eab}$.*

► **Proposition 13 ([4]).** *Let F be a (convex)¹ facet of \mathcal{S} . For any convex Borel set $R \subset F$, we have*

$$\left(\frac{a(R)}{4\pi\varepsilon^2}\right) \leq \#(R \cap \mathcal{X}) \leq \left(\frac{\kappa \cdot a(R \oplus \varepsilon)}{\pi\varepsilon^2}\right), \text{ and therefore,} \tag{1}$$

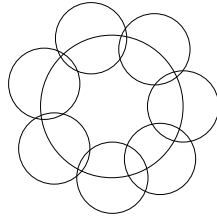
$$\left(\frac{A}{4\pi\varepsilon^2}\right) \leq \#(\mathcal{S} \cap \mathcal{X}) = n. \tag{2}$$

► **Proposition 14 ([4]).** *Let F be a facet of \mathcal{S} , let $\Gamma \subset F$ be a curve contained in F , and $k \in \mathbb{N}$. Then*

$$\#((\Gamma \oplus k\varepsilon) \cap \mathcal{X}) \leq \kappa \frac{l(\Gamma)(2k+2)\varepsilon}{\pi\varepsilon^2} \leq \left(2k\kappa \frac{l(\Gamma)}{\varepsilon}\right), \text{ when } k \geq 1. \tag{3}$$

► **Lemma 15.** *Given a circle $\Sigma_1 \subset \mathbb{E}^2$ of unit radius centered at the origin, seven disks having centers in Σ_1 and radius $1/2$, are necessary and sufficient to cover Σ_1 .*

► **Lemma 16 (Level Size).** $\#(L_{=k} \cap \mathcal{X}) \leq \#(L_{\leq k} \cap \mathcal{X}) \leq 9\kappa L \left(\frac{2^k \delta}{\varepsilon^2}\right)$.



► **Lemma 17.** *Let F be a facet of \mathcal{S} with supporting plane P , and $x \in F$ with $Lev(x) > 0$. Then given any $k \in [0, Lev(x))$, $k \in \mathbb{N}$, there exists a collection \mathcal{D}_x of at most $c_B = 7$ disks in F , such that*

- (i) *Each $D \in \mathcal{D}_x$ is contained in F ,*
- (ii) *Each $D \in \mathcal{D}_x$ has radius $r_0/4$, where $r_0 = 2^k \delta$ and $k \in \mathbb{N}$ such that $0 \leq k < Lev(x)$, and*
- (iii) *Any disk $D \subset P$ of radius at least r_0 , such that $x \in \partial D$, contains at least one disk in \mathcal{D}_x .*

¹ Recall our assumption in 2.4.

► **Lemma 18** (Decay Lemma). *Given $x_1, \dots, x_t \in \mathcal{X}$, with x_i contained in the facet F_i with supporting plane P_i , such that $\text{Lev}(x_i) > 0$, $1 \leq i \leq t$, then for all $0 \leq k_i < \text{Lev}(x_i)$, with $r_i^* := 2^{k_i} \delta$, the probability of the event*

$$E := \{\forall i \in [t] : \exists D_i = D(y_i, r_i) \subset P_i : r_i \geq r_i^*, x_i \in \mathcal{Y}, x_i \in \partial D_i \text{ and } \text{int}(D_i) \cap \mathcal{Y} = \emptyset\},$$

is given by

$$\mathbb{P}[E] \leq \begin{cases} q^t, & \text{if } k_{\max} = 0, \\ c_1 \cdot q^t \cdot \exp(-c_2 \cdot 2^{2k_{\max}}), & \text{if } k_{\max} > 0, \end{cases}$$

where $c_1 = c_B^t$, $c_2 \geq 2^{-7}$, and $k_{\max} := \max_i \{k_i\}$. Thus

$$\mathbb{P}[E] \leq c_1 \cdot q^t \cdot \exp(-c_2 \cdot 2^{2k_{\max}}), \quad k_{\max} \geq 0.$$

► **Lemma 19** (Growth Lemma). *Given any point $x \in \mathcal{S}$ in a facet F , and $0 \leq k < \text{Lev}(x)$, we have*

- (i) $2^{2k-2}/q \leq \#(D(x, 2^k \delta) \cap \mathcal{X}) \leq 4 \cdot (2^{2k}/q)$.
- (ii) $2^{2k-2} \leq \mathbb{E}[\#(D(x, 2^k \delta) \cap \mathcal{Y})] \leq 4 \cdot (2^{2k})$.

4.2 Proofs of Lemmas 8–11

The proofs of Lemmas 8 and 9 now follow by adapting the analysis of [4] to random subsamples of ε -nets, using the Decay and Growth lemmas.

Proof of Lemma 8. To bound the expected number of edges in E_1 , we simply bound the number of pairs $(x_1, x_2) \in \text{Bd}_{\mathcal{S}}(\mathcal{Y}) \times \text{Bd}_{\mathcal{S}}(\mathcal{Y})$. Consider a pair of points $x_1, x_2 \in \mathcal{X}$. Let $l_1 := \text{Lev}(x_1)$ and $l_2 := \text{Lev}(x_2)$, and let $l := \max\{l_1, l_2\}$. By definition, if $l = 0$, then $x_1, x_2 \in \text{Bd}_{\mathcal{S}}(\mathcal{Y})$. For $l \geq 1$, we get that $x_1 \in \text{Bd}_{\mathcal{S}}(\mathcal{Y})$ and $x_2 \in \text{Bd}_{\mathcal{S}}(\mathcal{Y})$ only if there exists a disk of radius at least $2^{l-1} \delta$ passing through x_1 or x_2 , and containing no points of \mathcal{Y} . Therefore to bound the probability that $(x_1, x_2) \in (\text{Bd}_{\mathcal{S}}(\mathcal{Y}))^2$, we can apply the Decay Lemma 18, with $t = 2$, for $i \in \{1, 2\}$. We get

$$\begin{aligned} \mathbb{P}[(x_1, x_2) \in E_1] &\leq \mathbb{P}[(x_1, x_2) \in (\text{Bd}_{\mathcal{S}}(\mathcal{Y}))^2] \\ &\leq c_1 q^2 \cdot \exp(-c_2 \cdot 2^{2l-2}) \leq c_1 q^2 \cdot \exp(-c'_2 \cdot 2^{2l}), \end{aligned} \quad (4)$$

where $c'_2 = c_2/4 = 2^{-9}$. Summing over all choices of levels of x_1 and x_2 , we have

$$\mathbb{E}[\#(E_1)] \leq \sum_{l_1 \geq 0} \#(L_{=l_1} \cap \mathcal{X}) \sum_{l_2 \geq 0} \#(L_{=l_2} \cap \mathcal{X}) \mathbb{P}[(x_1, x_2) \in (\text{Bd}_{\mathcal{S}}(\mathcal{Y}))^2].$$

By symmetry, it is enough to assume without loss of generality that $l_1 \geq l_2$, i.e. $l = l_1$. Thus,

$$\mathbb{E}[\#(E_1)] \leq 2 \sum_{l_1 \geq 0} \#(L_{=l_1} \cap \mathcal{X}) \sum_{l_2=0}^{l_1} \#(L_{=l_2} \cap \mathcal{X}) \mathbb{P}[(x_1, x_2) \in (\text{Bd}_{\mathcal{S}}(\mathcal{Y}))^2].$$

Applying equation (4) and the Level Size Lemma 16, we get

$$\begin{aligned} \mathbb{E}[\#(E_1)] &\leq 2 \sum_{l_1 \geq 0} \#(L_{\leq l_1} \cap \mathcal{X}) \sum_{l_2=0}^{l_1} \#(L_{\leq l_2} \cap \mathcal{X}) \cdot c_1 q^2 \cdot \exp(-c'_2 \cdot 2^{2l_1}) \\ &\leq 2c_1 q^2 \sum_{l_1 \geq 0} (9\kappa L \cdot (2^{l_1} \delta / \varepsilon^2)) \sum_{l_2=0}^{l_1} (9\kappa L \cdot (2^{l_2} \delta / \varepsilon^2)) \cdot \exp(-c'_2 \cdot 2^{2l_1}) \\ &\leq 2c_1 q^2 (9\kappa L \left(\frac{\delta}{\varepsilon^2}\right))^2 \sum_{l_1 \geq 0} 2^{l_1} \cdot \exp(-c'_2 \cdot 2^{2l_1}) \sum_{l_2=0}^{l_1} 2^{l_2}. \end{aligned}$$

22:10 RIC of Delaunay Triangulations of Nice Point Sets

Using the definitions of q and δ , together with Proposition 13, and writing the terms outside the summation as N_1 , we get $N_1 := 2 \cdot c_1 q^2 (9\kappa L (\frac{\delta}{\varepsilon^2}))^2 = 2c_1 \cdot (9\kappa L)^2 (\frac{s}{n\varepsilon^2}) \leq 4c_1 \cdot (\frac{4\pi(9\kappa L)^2}{A}) \cdot s$. We get

$$\mathbb{E} [\#(E_1)] \leq N_1 \sum_{l_1 \geq 0} 2^{l_1} \cdot \exp(-c'_2 \cdot 2^{2l_1}) \cdot 2 \cdot 2^{l_1} \leq 2N_1 \sum_{l_1 \geq 0} 2^{2l_1} \cdot \exp(-c'_2 \cdot 2^{2l_1}).$$

The summation can be bounded using Lemma 12, to get

$$\mathbb{E} [\#(E_1)] \leq 2N_1 \cdot \left(2 \cdot \frac{\log 1/c'_2}{2ec'_2} \right) = 2N_1 \cdot \frac{\log 1/c'_2}{e \cdot c'_2}.$$

Now substituting $c'_2 = 2^{-9}$ gives $\mathbb{E} [\#(E_1)] \leq 2 \cdot 10^4 \cdot c_1 \cdot (\frac{4\pi(9\kappa L)^2}{A}) \cdot s$. \blacktriangleleft

The proof of Lemma 9 follows simply from the fact that for any given face, the Delaunay graph formed by the points in \mathcal{Y} is planar, and therefore the number of edges is at most 3 times the number of points. The total number of such edges, summed over all faces of \mathcal{S} , is at most $3s$. The proof of Lemma 10 is based on combining a construction of Attali-Boissonnat with Lemma 9, and is omitted here. For the proofs of Lemmas 10 and 11, we need some more geometric ideas of [4]. Before proving Lemma 11, we briefly describe a construction, which will be central to our analysis.

► **Construction 20** (Attali-Boissonnat [4]). *Let P be a plane and Z be a finite set of points. To each point $x \in Z$, assign the region $V(x) = V_x(Z) \subset P$ of points $y \in P$ such that the sphere tangent to P at y and passing through x encloses no point of Z . Let $\mathcal{V} := \{V(x) : x \in Z\}$.*

We summarize some conclusions of Attali-Boissonnat regarding the construction. The proofs of these propositions can be found in [4].

► **Proposition 21.**

- (i) \mathcal{V} is a partition of P .
- (ii) For each $x \in Z$, $V(x)$ is an intersection of regions that are either disks or complements of disks.
- (iii) The total length of the boundary curves in \mathcal{V} is equal to the total length of the convex boundaries.

For the rest of this subsection, we shall apply Construction 20 on the plane P , and the points in $Bd_{\mathcal{S}}(\mathcal{Y})$ as Z . Let $\mathcal{T} := \text{Int}_F(\mathcal{Y})$ for some facet $F \in \mathcal{S}$. Given $x \in Z$, $y \in P \setminus V(x)$, let $k_y = k_y(x)$ denote the least $k \geq 0$ such that $y \in \partial V(x) \oplus 2^k \delta$.

► **Proposition 22** (Attali-Boissonnat [4]). *Suppose there exists a ball $B \subset \mathbb{E}^3$ and $y \in P$, such that $y, x \in \partial B$, and $B \cap \mathcal{T} = \emptyset$. Then the disk $D_y = P \cap B$ satisfies $D_y \cap \mathcal{T} = \emptyset$, $y \in \partial D_y$ and $D_y \cap V_x \neq \emptyset$.*

► **Lemma 23.** *If $\{x, y\} \in E_4$ with $x \in Bd_{\mathcal{S}}(\mathcal{Y})$, $y \in \text{Int}(F)$, then $k_y \leq \text{Lev}(y)$.*

Proof. Suppose $\{x, y\} \in E_4$. Then there exists a ball $B \in \mathbb{E}^3$ with $x, y \in \partial B$, and $\text{int}(B) \cap \mathcal{Y} = \emptyset$. Therefore $D_y := B \cap P$ also satisfies $\text{int}(D_y) \cap \mathcal{Y} = \emptyset$. By Proposition 22 we have that $D_y \cap V(x) \neq \emptyset$. Therefore, $y \in V(x) \oplus 2r_y$, where r_y is the radius of D_y . But since $y \in \text{Int}(F)$, we have that any disk having y on its boundary and containing no point of \mathcal{Y} in its interior can have radius at most $2^{\text{Lev}(y)-1} \delta$. Therefore $r_y \leq 2^{\text{Lev}(y)-1} \delta$. Now taking k_y such that $2^{k_y} \delta = 2r_y$, we get that $k_y \leq \text{Lev}(y)$. \blacktriangleleft

Now we partition the pairs of vertices $\{x, y\} \in E_4$ with $x \in Bd_S(\mathcal{Y})$, depending on whether $y \in V_F(x)$ or $y \in \partial V_F(x) \oplus 2^{k_y}\delta$. That is, given a facet $F \in \mathcal{S}$, let $E_4(Int(F))$ denote the set of edges $\{x, y\} \in E_4$ with $y \in int(V_F(x))$, and $E_4(Bd(F))$ denote the set of edges in E_4 with $y \in \partial(V_F(x)) \oplus 2^k\delta$, for $k \in [0, k_y]$. Define $E_4(Int) := \bigcup_{F \in \mathcal{S}} E_4(Int(F))$ and $E_4(Bd) := \bigcup_{F \in \mathcal{S}} E_4(Bd(F))$ respectively.

Lemma 11. The proof follows from Lemmas 24 and 25, which bound the expected number of edges in $E_4(Int)$ and $E_4(Bd)$ respectively. \blacktriangleleft

► **Lemma 24.** *Given a facet $F \in \mathcal{S}$, $\mathbb{E}[E_4(Int(F))] \leq q \cdot \#(\mathcal{X} \cap F)$. As a consequence, $\mathbb{E}[E_4(Int)] \leq s$.*

Proof. Let $x \in \mathcal{X}$ and $y \in \mathcal{X} \cap F$. Let $\mathcal{E}_{x,y}$ denote the event $\{x, y\} \in E_4(Int(F))$. Then $\mathcal{E}_{x,y}$ can occur only if (i) $x \in Bd_S(\mathcal{Y})$ and, (ii) $y \in Int_S(Y) \cap V_F(x)$. Fix a choice of \mathcal{Y} , say $Y \in \binom{\mathcal{X}}{s}$. Conditioning on this choice of \mathcal{Y} , $Bd_S(\mathcal{Y})$ is a fixed set of points. The number of pairs contributing to $E_4(Int(F))$ is at most $\#(\{(x, y) \in Y \times Y \mid x \in Bd_S(\mathcal{Y}), y \in V_F(x)\})$. The main observation is now that since \mathcal{V} restricted to F is a sub-division of F , for each $y \in \mathcal{X} \cap F$, there is a unique $x = x_y \in Bd_S(\mathcal{Y})$ such that $y \in V_F(x)$. Therefore we get

$$E_4(Int(F)) \leq \sum_{V_F(x) \in \mathcal{V}: x \in Bd_S(\mathcal{Y})} \#(V_F(x) \cap Y) \leq \#(Y \cap F).$$

Since the last bound holds for any choice of Y , taking expectation over all choices we get

$$\mathbb{E}[E_4(Int(F))] \leq \mathbb{E}[\#(\mathcal{Y} \cap F)] = q \cdot \#(\mathcal{X} \cap F).$$

Now summing over all faces gives $\mathbb{E}[E_4(Int)] \leq \mathbb{E}[\#(\mathcal{Y})] = s$. \blacktriangleleft

► **Lemma 25.** *Given a facet $F \in \mathcal{S}$, $\mathbb{E}[E_4(Bd(F))] \leq \left(O(1) \cdot \frac{\kappa^2 L \cdot l(\partial F) s}{A}\right)$. As a consequence, $\mathbb{E}[E_4(Bd(\mathcal{S}))] \leq \left(O(1) \cdot \frac{\kappa^2 L^2 s}{A}\right)$.*

Proof. To compute the expected value of $E_4(Bd(\mathcal{S}))$, fix a face $F \in \mathcal{S}$. Consider a pair of points $x, y \in \mathcal{X}$, such that $y \in F$. Let $\mathcal{E}_{x,y}$ denote the event $\{x, y\} \in E_4(Bd(F))$.

The value of $E_4(Bd)$ is the number of $x, y \in \mathcal{X}$, such that $\mathcal{E}_{x,y}$ occurs. Taking expectations,

$$\mathbb{E}[E_4(Bd(F))] \leq \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{X} \cap F} \mathbb{P}[\mathcal{E}_{x,y}]. \quad (5)$$

Observe that $\mathcal{E}_{x,y}$ occurs only if (i) $x \in Bd_S(\mathcal{Y})$ and (ii) $k_y(x) \leq Lev(y)$, by applying Construction 20, on the plane P , $Z = Bd_S(\mathcal{Y})$, and $\mathcal{T} = \mathcal{Y} \cap P$, and using Proposition 22. By Lemma 23, $k_y(x) \in [0, Lev(y)]$.

Let P_{l_1, l_2} denote the probability that $\{x, y\} \in E_4(Bd(F))$, with $Lev(x) = l_1$, and $k_y(x) = l_2$. Equation (5) can be rewritten in terms of l_1 and l_2 as

$$\mathbb{E}[E_4(Bd(F))] \leq \sum_{l_1 \geq 0} \#(L_{=l_1} \cap \mathcal{X}) \sum_{l_2 \geq 0} \sum_{V_F \in \mathcal{V}} \#((\partial V_F \oplus 2^{l_2}\delta) \cap \mathcal{X}) \cdot P_{l_1, l_2}.$$

Applying the Decay Lemma 18 with $t = 2$, $x_1 = x$, $x_2 = y$, $k_1 = \max\{0, l_1 - 1\}$ (since $x \in Bd_S(\mathcal{Y})$), and $k_2 = \max\{0, l_2 - 1\}$, we get

$$P_{l_1, l_2} \leq c_1 q^2 \cdot \exp(-f(l^*)),$$

where $l^* := \max\{0, l_1 - 1, l_2 - 1\}$, and $f(l^*) = 0$ if $l^* = 0$, and $c'_2 \cdot 2^{2l^*}$ otherwise, with $c'_2 = c_2/4$. As in the proof of Lemma 8, we shall use symmetry to combine the cases $l_1 \geq l_2$ and $l_2 > l_1$ together.

$$\begin{aligned} \mathbb{E}[E_4(Bd(F))] &\leq 2 \sum_{l_1 \geq 0} \#(L_{=l_1} \cap \mathcal{X}) \cdot \\ &\quad \sum_{l_2 \leq l_1} \sum_{V(x) \in \mathcal{V}} \#((\partial V(x) \oplus 2^{l_2} \delta) \cap \mathcal{X}) \cdot c_1 q^2 \cdot \exp(-c'_2 2^{2l_1}). \end{aligned}$$

By the Level Size Lemma 16, we get that $\#(L_{=l_1} \cap \mathcal{X}) \leq \frac{2\kappa L 2^{l_1} \delta}{\varepsilon^2}$. Using Proposition 14, we get that $\#(\{\partial V(x) \oplus 2^{l_2} \delta\} \cap \mathcal{Y}) \leq \frac{2\kappa \cdot l(\partial V(x)) 2^{l_2} \delta}{\varepsilon^2}$. By Proposition 21 (iii), each boundary in the partition \mathcal{V} is convex for some $x \in Bd_S(\mathcal{Y})$, and so we need to sum $l(\partial V(x))$ only over the convex curves in $\partial V(x)$, $x \in Bd_S(\mathcal{Y})$, whose length we observe is at most $l(\partial F)$. Thus,

$$\mathbb{E}[E_4(Bd(F))] \leq 2L \cdot l(\partial F) \cdot \left(\frac{2\kappa \delta q}{\varepsilon^2}\right)^2 \sum_{l_1 \geq 0} 2^{l_1} \sum_{l_2 \leq l_1} 2^{l_2} c_1 \cdot \exp(-c'_2 \cdot 2^{2l_1}).$$

Using Lemma 12, the above summation is bounded by a constant. This comes to $c_1 \cdot O(1) \left(\frac{(2\kappa)^2 L \cdot l(\partial F) \delta^2 q^2}{\varepsilon^4}\right) = O\left(\frac{\kappa^2 L \cdot l(\partial F) s}{A}\right)$, where the last step followed from the lower bound on n in Proposition 13 (2), and the identities $q = s/n = \delta^2/\varepsilon^2$. Summing y over all facets F in \mathcal{S} , we get $\mathbb{E}[E_4(Bd(\mathcal{S}))] = \left(O(1) \cdot \frac{\kappa^2 L^2 s}{A}\right)$. ◀

References

- 1 N. Amenta and M. Bern. Surface Reconstruction by Voronoi Filtering. *Discrete & Computational Geometry*, 22(4):481–504, December 1999.
- 2 Nina Amenta, Dominique Attali, and Olivier Devillers. Complexity of Delaunay triangulation for points on lower-dimensional polyhedra. In *18th ACM-SIAM Symposium on Discrete Algorithms*, pages 1106–1113, 2007.
- 3 Nina Amenta, Sunghee Choi, and Günter Rote. Incremental constructions con BRIO. In *Proc. 19th Annual Symposium on Computational geometry*, pages 211–219, 2003. doi:10.1145/777792.777824.
- 4 Dominique Attali and Jean-Daniel Boissonnat. A Linear Bound on the Complexity of the Delaunay Triangulation of Points on Polyhedral Surfaces. *Discrete & Computational Geometry*, 31(3):369–384, February 2004.
- 5 Dominique Attali, Jean-Daniel Boissonnat, and André Lieutier. Complexity of the Delaunay Triangulation of Points on Surfaces: the Smooth Case. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, SCG '03, pages 201–210, New York, NY, USA, 2003. ACM. doi:10.1145/777792.777823.
- 6 Jean-Daniel Boissonnat and Frédéric Cazals. Smooth surface reconstruction via natural neighbour interpolation of distance functions. *Computational Geometry*, 22(1):185–203, 2002. 16th ACM Symposium on Computational Geometry.
- 7 Jean-Daniel Boissonnat, Olivier Devillers, Kunal Dutta, and Marc Glisse. Randomized incremental construction of Delaunay triangulations of nice point sets. preprint, December 2018. URL: <https://hal.inria.fr/hal-01950119>.
- 8 Jean-Daniel Boissonnat, Olivier Devillers, and Samuel Hornus. Incremental Construction of the Delaunay Triangulation and the Delaunay Graph in Medium Dimension. In *Proceedings of the Twenty-fifth Annual Symposium on Computational Geometry*, SCG '09, pages 208–216, New York, NY, USA, 2009. ACM. doi:10.1145/1542362.1542403.
- 9 Jean-Daniel Boissonnat and Monique Teillaud. On the randomized construction of the Delaunay tree. *Theoretical Computer Science*, 112:339–354, 1993. doi:10.1016/0304-3975(93)90024-N.

- 10 Keneth L. Clarkson and Peter W. Shor. Applications of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4:387–421, 1989. doi:10.1007/BF02187740.
- 11 Olivier Devillers. Randomization Yields Simple $O(n \log^* n)$ Algorithms for Difficult $\Omega(n)$ Problems. *International Journal of Computational Geometry and Applications*, 2(1):97–111, 1992. URL: <https://hal.inria.fr/inria-00167206>.
- 12 Olivier Devillers. The Delaunay hierarchy. *International Journal of Foundations of Computer Science*, 13:163–180, 2002. hal:inria-00166711.
- 13 R.A. Dwyer. The expected number of k-faces of a Voronoi diagram. *Computers & Mathematics with Applications*, 26(5):13–19, 1993.
- 14 Rex A. Dwyer. Higher-dimensional Voronoi diagrams in linear expected time. *Discrete & Computational Geometry*, 6(3):343–367, September 1991.
- 15 Jeff Erickson. Nice Point Sets Can Have Nasty Delaunay Triangulations. In *Proceedings of the Seventeenth Annual Symposium on Computational Geometry*, SCG '01, pages 96–105, New York, NY, USA, 2001. ACM.
- 16 Jeff Erickson. Dense Point Sets Have Sparse Delaunay Triangulations. *Discrete & Computational Geometry*, 33(1):83–115, January 2005.
- 17 Mordecai J. Golin and Hyeon-Suk Na. On the average complexity of 3D-Voronoi diagrams of random points on convex polytopes. *Computational Geometry*, 25(3):197–231, 2003.
- 18 J. L. Meijering. Interface area, edge length, and number of vertices in crystal aggregates with random nucleation. *Philips Research Reports*, 8:270–290, 1953.
- 19 J. Møller. Random tessellations in \mathbb{R}^d . *Advances in Applied Probability*, 21(1):37–73, 1989.
- 20 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.
- 21 Ketan Mulmuley. *Computational geometry - an introduction through randomized algorithms*. Prentice Hall, 1994.
- 22 The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.14 edition, 2019. URL: <https://doc.cgal.org/4.14/Manual/packages.html>.

Fine-Grained Complexity of k -OPT in Bounded-Degree Graphs for Solving TSP

Édouard Bonnet

ENS Lyon, LIP, Lyon, France
edouard.bonnet@ens-lyon.fr

Yoichi Iwata

National Institute of Informatics, Tokyo, Japan
yiwata@nii.ac.jp

Bart M. P. Jansen 

Eindhoven University of Technology, Eindhoven, The Netherlands
b.m.p.jansen@tue.nl

Łukasz Kowalik 

Institute of Informatics, University of Warsaw, Poland
kowalik@mimuw.edu.pl

Abstract

The TRAVELING SALESMAN PROBLEM asks to find a minimum-weight Hamiltonian cycle in an edge-weighted complete graph. Local search is a widely-employed strategy for finding good solutions to TSP. A popular neighborhood operator for local search is k -opt, which turns a Hamiltonian cycle \mathcal{C} into a new Hamiltonian cycle \mathcal{C}' by replacing k edges. We analyze the problem of determining whether the weight of a given cycle can be decreased by a k -opt move. Earlier work has shown that (i) assuming the Exponential Time Hypothesis, there is no algorithm that can detect whether or not a given Hamiltonian cycle \mathcal{C} in an n -vertex input can be improved by a k -opt move in time $f(k)n^{o(k/\log k)}$ for any function f , while (ii) it is possible to improve on the brute-force running time of $\mathcal{O}(n^k)$ and save linear factors in the exponent. Modern TSP heuristics are very successful at identifying the *most promising* edges to be used in k -opt moves, and experiments show that very good global solutions can already be reached using only the top- $\mathcal{O}(1)$ most promising edges incident to each vertex. This leads to the following question: can improving k -opt moves be found efficiently in graphs of bounded degree? We answer this question in various regimes, presenting new algorithms and conditional lower bounds. We show that the aforementioned ETH lower bound also holds for graphs of maximum degree three, but that in bounded-degree graphs the best improving k -move can be found in time $\mathcal{O}(n^{(23/135+\epsilon_k)k})$, where $\lim_{k \rightarrow \infty} \epsilon_k = 0$. This improves upon the best-known bounds for general graphs. Due to its practical importance, we devote special attention to the range of k in which improving k -moves in bounded-degree graphs can be found in *quasi-linear* time. For $k \leq 7$, we give quasi-linear time algorithms for general weights. For $k = 8$ we obtain a quasi-linear time algorithm when the weights are bounded by $\mathcal{O}(\text{polylog } n)$. On the other hand, based on established fine-grained complexity hypotheses about the impossibility of detecting a triangle in edge-linear time, we prove that the $k = 9$ case does not admit quasi-linear time algorithms. Hence we fully characterize the values of k for which quasi-linear time algorithms exist for polylogarithmic weights on bounded-degree graphs.

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis; Theory of computation \rightarrow Parameterized complexity and exact algorithms

Keywords and phrases traveling salesman problem, k -OPT, bounded degree

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.23

Related Version A full version of the paper is available at <http://arxiv.org/abs/1908.09325>.

Funding Yoichi Iwata: Supported by JSPS KAKENHI Grant Number JP17K12643.

Bart M. P. Jansen: Supported by ERC Starting Grant ReduceSearch (Grant Agreement No 803421).

Łukasz Kowalik: Supported by ERC Starting Grant TOTAL (Grant Agreement No 677651).



© Édouard Bonnet, Yoichi Iwata, Bart M. P. Jansen, and Łukasz Kowalik;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 23; pp. 23:1–23:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements This research was initiated at the Shonan Meeting *Parameterized Graph Algorithms & Data Reduction: Theory Meets Practice* held during March 4–8, 2019 in Shonan Village Center, Japan. Yoichi Iwata thanks the Kaggle Traveling Santa 2018 Competition for motivating him to study practical TSP heuristics. He also thanks Shogo Murai for valuable discussion about the possibility of faster k -OPT algorithms.

1 Introduction

1.1 Motivation

The TRAVELING SALESMAN PROBLEM (TSP) hardly needs an introduction; it is one of the most important problems in combinatorial optimization, which asks to find a Hamiltonian cycle of minimum weight in an edge-weighted complete graph. Local search is widely used in practical TSP solvers [10, 11]. The most commonly used neighborhood is a k -move (or k -opt move). A k -move on a Hamiltonian cycle \mathcal{C} is a pair (E^-, E^+) of edge sets such that $E^- \subseteq E(\mathcal{C})$, $|E^-| = |E^+| = k$ and $(\mathcal{C} \setminus E^-) \cup E^+$ is also a Hamiltonian cycle. Marx [13] showed that finding an improving k -move (i.e., a k -move that results in a lighter Hamiltonian cycle) is W[1]-hard parameterized by k , and this result was refined by Guo et al. [6] to obtain an $f(k)n^{\Omega(k/\log k)}$ lower bound under the Exponential Time Hypothesis (ETH). For small values of k , the current fastest running time is $\mathcal{O}(n^k)$ for $k = 2, 3$ (by exhaustive search), $\mathcal{O}(n^3)$ for $k = 4$ [4], and $\mathcal{O}(n^{3.4})$ for $k = 5$ [3]. Moreover, de Berg et al. [4] and Cygan et al. [3] showed that improving the running time to $\mathcal{O}(n^{3-\epsilon})$ for $k = 3$ or $k = 4$ implies a breakthrough result of an $\mathcal{O}(n^{3-\delta})$ -time algorithm for ALL-PAIRS SHORTEST PATHS.

From the hardness shown by the theoretical studies, it seems that local search can be applied only to small graphs. Nevertheless, state-of-the-art local search TSP solvers can deal with large graphs with tens of thousands of vertices. This is mainly due to the following two heuristics.

1. They sparsify the input graph by picking the top- d important incident edges for each vertex according to an appropriate importance measure. For example, Lin-Kernighan [12] picks the top-5 nearest neighbors, and its extension LKH [8] picks the top-5 α -nearest neighbors, where the α -distance of an edge is the increase of the Held-Karp lower bound [7] by including the edge. The empirical evaluation by Helsgaun [8] showed that the sparsification by the α -nearest neighbors can preserve almost optimal solutions.
2. They mainly focus on *sequential* k -moves. In general, $E^- \cup E^+$ is a set of edge-disjoint closed walks, each of which alternately uses edges in E^- and E^+ . If it consists of a single closed walk, the move is called sequential. Graphs of maximum degree d with n vertices have at most $n(2(d-2))^{k-1}$ sequential k -moves (n choices for the starting point, 2 choices for the next edge in E^- , and at most $d-2$ choices for the next edge in E^+), which is linear in n when considering d and k as constants. On the other hand, linear-time computation of non-sequential k -moves appears non-trivial. Lin-Kernighan does not search for non-sequential moves at all, and after it finds a local optimum, it applies special non-sequential 4-moves called *double bridges* to get out of the local optimum. LKH-2 [9] improves Lin-Kernighan by heuristically searching for non-sequential moves during the local search.

This state of affairs raises the following questions: what is the complexity of finding improving k -moves in bounded-degree graphs? How does the complexity scale with k , and can it be done efficiently for small values of k ? Since improving *sequential moves* can be found in linear time for fixed k and d , to answer these questions we have to investigate non-sequential k -moves in bounded-degree graphs.

1.2 Our contributions

We classify the complexity of finding improving k -moves in bounded-degree graphs in various regimes. We present improved algorithms that exploit the degree restrictions using the structure of k -moves, treewidth bounds, color-coding, and suitable data structures. We also give new lower bounds based on the Exponential Time Hypothesis (ETH) and hypotheses from fine-grained complexity concerning the complexity of detecting triangles. To state our results in more detail, we first introduce the two problem variants we consider; a weak variant to which our lower bounds already apply, and a harder variant which can be solved by our algorithms.

k -OPT DETECTION

Parameter: k .

Input: An undirected graph G , a weight function $w: E(G) \rightarrow \mathbb{Z}$, an integer k , and a Hamiltonian cycle $\mathcal{C} \subseteq E(G)$.

Question: Can \mathcal{C} be changed into a Hamiltonian cycle of strictly smaller weight by a k -move?

The related optimization problem k -OPT OPTIMIZATION is to compute, given a Hamiltonian cycle in the graph, a k -move that gives the largest cost improvement, or report that no improving k -move exists. With this terminology, we describe our results.

We show that k -OPT DETECTION is unlikely to be fixed-parameter tractable on bounded-degree graphs: we give a new constant-degree lower-bound construction to show that there is no function f for which k -OPT DETECTION on subcubic graphs with weights $\{1, 2\}$ can be solved in time $f(k) \cdot n^{o(k/\log k)}$, unless ETH fails. Hence the running time lower bound for general graphs by Guo et al. [6] continues to hold in this very restricted setting. While the degree restriction does not make the problem fixed-parameter tractable, it is possible to obtain faster algorithms. By adapting the approach of Cygan et al. [3], exploiting the fact that the number of sequential moves is linear in n in bounded-degree graphs, and proving a new upper bound on the pathwidth of an k -edge even graph, we show that k -OPT OPTIMIZATION in n -vertex graphs of maximum degree $\mathcal{O}(1)$ can be solved in time $\mathcal{O}(n^{(23/135+\epsilon_k)k}) = \mathcal{O}(n^{(0.1704+\epsilon_k)k})$, where $\lim_{k \rightarrow \infty} \epsilon_k = 0$. This improves on the behavior for general graphs, where the current-best running time [3] is $\mathcal{O}(n^{(1/4+\epsilon_k)k})$.

Since quasi-linear running times are most useful for dealing with large inputs, we perform a fine-grained analysis of the range of k for which improving k -moves can be found in time $\mathcal{O}(n \text{ polylog } n)$ on n -vertex graphs. Observe that in the bounded-degree setting, the number of edges m is $\mathcal{O}(n)$. We prove lower bounds using the hypothesis that detecting a triangle in an unweighted graph cannot be done in nearly-linear time in the number of edges m , which was formulated in several ways by Abboud and Vassilevska Williams [1, Conjectures 2–3]. By an efficient reduction from TRIANGLE DETECTION, we show that an algorithm with running time $\mathcal{O}(n \text{ polylog } n)$ for 9-OPT DETECTION in subcubic graphs with weights $\{1, 2\}$ implies that a triangle in an m -edge graph can be found in time $\mathcal{O}(m \text{ polylog } m)$, contradicting popular conjectures. We complement these lower bounds by quasi-linear algorithms for all $k \leq 8$ to obtain a complete dichotomy for the case of integer weights bounded by $\mathcal{O}(\text{polylog } n)$. When the weights are not bounded, we obtain quasi-linear time algorithms for all $k \leq 7$, leaving open only the case $k = 8$.

1.3 Organization

Preliminaries are presented in Section 2. In Section 3 we give faster XP algorithms for varying k . By refining these ideas, we give quasi-linear-time algorithms for $k \leq 8$ in Section 4. Section 5 gives the reduction from TRIANGLE DETECTION to establish a superlinear lower bound on subcubic graphs for $k = 9$. In Section 6 we describe the lower bound for varying k .

2 Preliminaries

Given a graph G edge-weighted by $w: E(G) \rightarrow \mathbb{Z}$, and a subset $F \subseteq E(G)$ of its edges, $w(F) := \sum_{e \in F} w(e)$. A k -move on a Hamiltonian cycle \mathcal{C} is pair (E^-, E^+) of edge sets such that $|E^-| = |E^+| = k$ and $(\mathcal{C} \setminus E^-) \cup E^+$ is also a Hamiltonian cycle. A k -move is called *improving* if $w((\mathcal{C} \setminus E^-) \cup E^+) < w(\mathcal{C})$, or equivalently and more simply $w(E^+) < w(E^-)$. A necessary condition for a pair (E^-, E^+) to be a k -move is that the multiset of endpoints of E^- is equal to the multiset of endpoints of E^+ . An exchange (E^-, E^+) that satisfies this condition is called a k -swap. We say that a k -swap *results* in the graph $(\mathcal{C} \setminus E^-) \cup E^+$. Note that a k -swap always results in a spanning disjoint union of cycles. A k -swap resulting in a graph with a single connected component is therefore a k -move. An *infeasible* k -swap is a k -swap which is not a k -move.

We say that a k -swap (E^-, E^+) *induces* the graph $E^- \cup E^+$. As a slight abuse of notation, a k -swap will sometimes directly refer to this graph. A k -swap (E^-, E^+) such that all edges $E^- \cup E^+$ are visited by a single closed walk alternating between E^- and E^+ is called *sequential*. In particular, in a simple graph, every 2-swap is sequential. One can notice that an infeasible (sequential) 2-swap results in a disjoint union of exactly two cycles. A k -move can always be decomposed into sequential k_i -swaps (with $\sum k_i = k$) but some k -moves cannot be decomposed into sequential k_i -moves. The quantity $w(E^-) - w(E^+)$ is called the *gain* of the swap (E^-, E^+) . We distinguish *neutral* swaps, with gain 0, *improving* swaps, with strictly positive gain, and *worsening* swaps, with strictly negative gain.

For an integer n , we denote $[n] = \{1, \dots, n\}$. A k -embedding (or shortly: *embedding*) is an increasing function $f: [k] \rightarrow [n]$. A *connection k -pattern* (or shortly: *connection pattern*) is a perfect matching in the complete graph on the vertex set $[2k]$. A pair (f, M) where f is a k -embedding and M is a connection k -pattern, is an alternative description of a k -swap. Indeed, let e_1, \dots, e_n be subsequent edges of \mathcal{C} . Then, $E^- = \{e_{f(i)}: i \in [k]\}$. Vertices of the connection pattern correspond to endpoints of E^- , i.e., vertices $2i - 1, 2i \in [2k]$ correspond to the left and right (in the clockwise order) endpoint of $e_{f(i)}$, respectively. Thus, edges of the connection pattern correspond to a set E^+ of $|M|$ edges in G . We say that a k -swap (E^-, E^+) *fits into* M if there is an embedding f such that (f, M) describes (E^-, E^+) . Note that every pair of an embedding and a connection pattern (f, M) describes exactly one swap (E^-, E^+) . Conversely, for a swap (E^-, E^+) the corresponding embedding f is also unique (and determined by E^-). However, in case E^- contains incident edges, the swap fits into more than one matching M (see Fig. 1). See [3] for a more formal description of the equivalence.

The notion of a connection pattern can be extended to represent k' -swaps, for $k' < k$, as follows. Note that a matching N in the complete graph on the vertex set $[2k]$ corresponds to an $|N|$ -swap if and only if there is a set $\iota(N) \subseteq [k]$ such that $V(N) = \{2i - 1, 2i: i \in \iota(N)\}$. For a set $X \subseteq [k]$, by $M[X]$ we denote the swap N such that $\iota(N) = X$. We say that a connection pattern M *decomposes* into swaps N_1, \dots, N_t when $M = \bigsqcup_{i=1}^t N_i$ and each N_i is a connection pattern of a swap. The notion of fitting extends to k' -swaps in the natural way.

Consider a connection pattern N of a swap, for $V(N) \subseteq [2k]$. We call N *sequential* if $N \cup \{\{2i - 1, 2i\}: i \in \iota(N)\}$ forms a simple cycle. In particular, every connection pattern can be decomposed into sequential connection patterns of (possibly shorter) swaps. The correspondence between sequential swaps and sequential connection patterns is somewhat delicate, so let us explain it in detail.

Let N be a sequential connection pattern, $V(N) \subseteq [2k]$. Recall that for every embedding f there is exactly one $|N|$ -swap (E^-, E^+) that fits into N . Clearly, this swap is sequential, since every edge in $\{\{2i - 1, 2i\}: i \in \iota(N)\}$ corresponds to an edge of E^- and every edge in

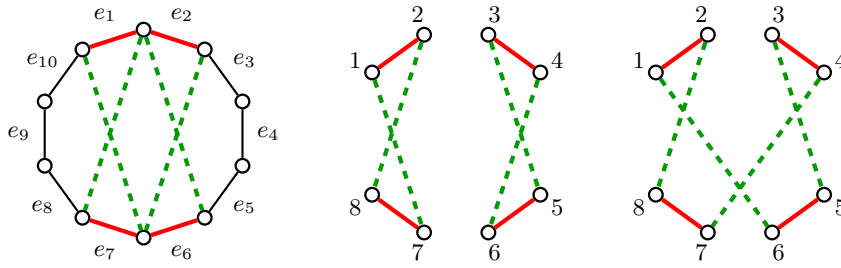


Figure 1 A sequential swap (left) which fits two connection patterns (center, right). The pattern in the center is not sequential, while the pattern on the right is sequential. On the left the solid red edges are in E^- , the dashed green edges are in E^+ , and the thin black edges are the remaining edges of the Hamiltonian cycle C . In the central and right pictures, the dashed green edges form some connection patterns.

N corresponds to an edge in E^+ . Thus the resulting set of edges $E^- \cup E^+$ forms a single closed walk. In particular, if the image of f contains two neighboring indices $i, i + 1 \in [n]$, the closed walk is not a simple cycle.

Conversely, it is possible that a sequential swap fits into a connection pattern which is not sequential, see Fig. 1 for an example. However, every sequential ℓ -swap (E^-, E^+) fits at least one sequential connection pattern. This sequential connection pattern is determined by the closed walk which certifies the sequentiality of the swap. Indeed, let $E^- = \{e_{i_1}, \dots, e_{i_\ell}\}$, where i_1, \dots, i_ℓ is an increasing sequence. Let $v_0, \dots, v_{2\ell-1}$ be the closed walk alternating between E^- and E^+ , in particular assume that $E^- = \{v_i v_{i+1} : i \text{ is even}\}$. Consider any $i = 0, \dots, \ell - 1$ and the corresponding edge $e_{i_j} = v_{2i} v_{2i+1}$ in E^- , for some $j \in [\ell]$. If v_{2i} is the left endpoint of e_{i_j} , we put $w_{2i} = 2j - 1$ and $w_{2i+1} = 2j$, otherwise $w_{2i} = 2j$ and $w_{2i+1} = 2j - 1$. Then $w_0, \dots, w_{2\ell-1}$ is a simple cycle and $N = \{w_i w_{i+1} : i \text{ is odd}\}$ is a sequential connection pattern. By construction, (E^-, E^+) fits N , as required. Keeping in mind the nuances in the notions of sequential swaps and corresponding sequential connection patterns, for simplicity, we will often just say “a sequential swap M ” for a matching M , instead of the more formal “a sequential connection pattern M of a swap”.

Fix a connection pattern M and let $f: S \rightarrow [n]$ be a partial embedding, for some $S \subseteq [k]$. For every $j \in S$, let v_{2j-1} and v_{2j} be the left and right endpoint of $e_{f(j)}$, respectively. We define

$$E_f^- = \{e_{f(i)} \mid i \in S\},$$

$$E_f^+ = \{\{v_{i'}, v_{j'}\} \mid i, j \in S, i' \in \{2i - 1, 2i\}, j' \in \{2j - 1, 2j\}, \{i', j'\} \in M\}.$$

Then, $\text{gain}_M(f) = w(E_f^-) - w(E_f^+)$.

3 Fast XP algorithms

For all fixed integers k and d , the number of sequential k -swaps in a graph of maximum degree d is $\mathcal{O}(n)$, and we can enumerate all of them in the same running time. Therefore, we can find the best improving k -move that can be decomposed into at most c sequential k -swaps in $\mathcal{O}(n^c)$ time. Because c is at most $\lfloor \frac{k}{2} \rfloor$, we obtain an $\mathcal{O}(n^{\lfloor \frac{k}{2} \rfloor})$ -time algorithm for k -OPT OPTIMIZATION. In what follows, we will improve this naive algorithm. Below we present a relatively simple algorithm which exploits the range tree data structure [15] and achieves running time roughly the same as the more sophisticated algorithm of Cygan et al. [3] for general graphs.

► **Theorem 1.** *For all fixed integers k , c , and d , there is an $\mathcal{O}(n^{\lceil \frac{c}{2} \rceil})$ polylog n -time algorithm to compute the best improving k -move that can be decomposed into c sequential swaps in graphs of maximum degree d .*

Proof. When $c = 1$, we can use the naive algorithm. Suppose $c \geq 2$ and let $h := \lceil \frac{c}{2} \rceil$.

For each possible connection pattern M consisting of c sequential swaps, we find the best embedding as follows. Let $M = \bigcup_{i=1}^c N_i$, where each N_i corresponds to a sequential swap. We split M into two parts $M_L = \bigcup_{i=1}^h N_i$ and $M_R = \bigcup_{i=h+1}^c N_i$ and we define $L = \bigcup_{i=1}^h \iota(N_i)$ and $R = \bigcup_{i=h+1}^c \iota(N_i)$. Note that $L \uplus R = [k]$. Let $f_L: L \rightarrow [n]$ and $f_R: R \rightarrow [n]$ be embeddings of L and R , respectively. The union of these two embeddings results in an embedding of $[k]$ if and only if the following conditions hold.

- For each $i \in [k-1]$ with $i \in L$ and $i+1 \in R$, $f_L(i) < f_R(i+1)$ holds.
- For each $i \in [k-1]$ with $i \in R$ and $i+1 \in L$, $f_R(i) < f_L(i+1)$ holds.

We can efficiently compute a pair of embeddings satisfying these conditions using an orthogonal range maximum data structure as follows. Let $\{l_1, \dots, l_p\} = \{i: l_i \in L \text{ and } l_i + 1 \in R\}$ and let $\{r_1, \dots, r_q\} = \{i: r_i - 1 \in R \text{ and } r_i \in L\}$. We first enumerate all the $|L|$ -swaps that fit into M_L and all the $|R|$ -swaps that fit into M_R , in $\mathcal{O}(n^h)$ time. For each such $|L|$ -swap (f_L, M_L) , we create a $(p+q)$ -dimensional point $(f_L(l_1), \dots, f_L(l_p), f_L(r_1), \dots, f_L(r_q))$ with a priority gain $_{M_L}(f_L)$, and we collect these points into a data structure. It stores $\mathcal{O}(n^h)$ points. For each $|R|$ -swap (f_R, M_R) , we query for the embedding f_L of maximum priority satisfying $f_L(l_i) < f_R(l_i + 1)$ for every $i \in [p]$ and $f_R(r_i - 1) < f_L(r_i)$ for every $i \in [q]$, and we answer the pair maximizing the total gain, i.e., the sum $\text{gain}_{M_L}(f_L) + \text{gain}_{M_R}(f_R)$. Using the range tree data structure [15], each query takes $\mathcal{O}(\log^{p+q} n^h) = \mathcal{O}(\text{polylog } n)$ time, so the total running time is $\mathcal{O}(n^h \text{ polylog } n)$. ◀

Since $c \leq \lfloor \frac{k}{2} \rfloor$ we get the following corollary.

► **Corollary 2.** *For all fixed integers k and d , k -OPT OPTIMIZATION in graphs of maximum degree d can be solved in time $\mathcal{O}(n^{\lceil \frac{k-1}{4} \rceil})$ polylog n .*

Let us take another look at the proof of Theorem 1. Recall that for merging embeddings f_L and f_R , we were interested only in values $f_L(i)$ for $i \in L$ such that $i+1 \in R$ or $i-1 \in R$. The embeddings of the remaining elements of L were forgotten at that stage, but we knew that it is possible to embed them and we stored the gain of embedding them. This suggests the following, different approach. We decompose the connection pattern into sequential swaps and we scan the swaps in a carefully chosen order. Assume we scanned t swaps already and there are $c-t$ swaps ahead. Assume that only $p \ll t$ of the t “boundary” swaps interact with the remaining $c-t$ swaps, where two swaps N_1 and N_2 interact when there is $i \in \iota(N_1)$ such that $i-1 \in \iota(N_2)$ or $i+1 \in \iota(N_2)$. Then it suffices to compute, for every embedding f_L of the p swaps, the gain of the best (i.e., giving the highest gain) embedding g_L of the t swaps, such that f_L matches g_L on the boundary swaps. This amounts to $\mathcal{O}(n^p)$ values to compute, since each sequential swap can be embedded in $\mathcal{O}(n)$ ways, if k and the maximum degree are $\mathcal{O}(1)$. The idea is to (1) compute these values quickly (in time linear in their number) using analogous values computed for the prefix of $t-1$ swaps, (2) find an order of swaps so that p is always small, namely $p \leq (23/135 + \epsilon_k)k$. The readers familiar with the notion of pathwidth recognize that p here is just the pathwidth of the graph obtained from the path $1, 2, \dots, k$ by identifying vertices in the set $\iota(N)$ for every sequential swap N in M , and that (2) is just dynamic programming over the path decomposition. The resulting algorithm is summarized in Theorem 3, and due to space limits, its formal proof is deferred to the full version.

► **Theorem 3.** *For all fixed integers k and d , k -OPT OPTIMIZATION in graphs of maximum degree d can be solved in time $\mathcal{O}(n^{(23/135+\epsilon_k)k}) = \mathcal{O}(n^{(0.1704+\epsilon_k)k})$, where $\lim_{k \rightarrow \infty} \epsilon_k = 0$.*

4 Fast algorithms for small k

Note that the algorithm for k -OPT OPTIMIZATION from Corollary 2 is quasi-linear for $k \leq 5$. In this section we extend the quasi-linear-time solvability to $k \leq 7$ for k -OPT DETECTION. Under an additional assumption of bounded weights, we are able to reach quasi-linear time for $k = 8$ as well, but the details of this part are deferred to the full version because of space constraints. To be precise, in the $k = 7$ case we prove the following stronger statement than just finding an arbitrary improving k -move.

► **Theorem 4.** *For $k \leq 7$, there is a quasi-linear-time algorithm to compute the best improving k -move in bounded-degree graphs under the assumption that there are no improving k' -moves for $k' < k$.*

We say that a connection pattern M of k -swaps is *reducible* if it can be decomposed into two moves. Note that if M is improving, then at least one of the two moves is improving, contradicting the assumption of Theorem 4.

► **Observation 5.** *If there are no improving k' -moves for $k' < k$, then no improving k -swap fits into a reducible connection pattern.*

Before we formulate our algorithm, we need two lemmas. We can prove these lemmas by case analysis, and because of the space constraints, their proofs are deferred to the full version. Let $M[X]$ and $M[Y]$ be two swaps in a connection pattern M , for some disjoint $X, Y \subseteq [k]$. *Interaction* between $M[X]$ and $M[Y]$ is any $i \in [k-1]$ such that $i \in X$ and $i+1 \in Y$ or $i \in Y$ and $i+1 \in X$.

► **Lemma 6.** *For any $k \geq 6$, there is no feasible and irreducible connection k -pattern that contains two 2-swaps that interact at least twice.*

Let M be a connection pattern, i.e., a perfect matching on vertices $[2k]$. We say that M' is obtained from M by swapping i and $i+1$, for $i \in [k]$, when M' is obtained from M by swapping the mates of $2i-1$ and $2i+1$ and swapping the mates of $2i$ and $2i+2$.

► **Lemma 7.** *Let M be a feasible irreducible connection k -pattern. Assume that M decomposes into three sequential swaps $M[X]$, $M[Y]$, and $M[Z]$, such that $|X| = |Y| = 2$. If there is exactly one index $i \in [k-1]$ with $i \in X$ and $i+1 \in Y$ or $i \in Y$ and $i+1 \in X$, the connection pattern M' obtained from M by swapping i and $i+1$ is either feasible or reducible.*

Now we are ready to describe the algorithm from Theorem 4 (see also Pseudocode 1). For each feasible and irreducible connection k -pattern M , we compute the best embedding as follows. If M consists of at most two sequential swaps, we can use the algorithm in Theorem 1. Otherwise, M consists of three sequential swaps $M[X]$, $M[Y]$, $M[Z]$ such that $X \uplus Y \uplus Z = [k]$, $|X| = |Y| = 2$ and $|Z| = k-4$. For each embedding $f_X : X \rightarrow [n]$ of $X = \{i, j\}$ we create a 2-dimensional point $(f_X(i), f_X(j))$ with priority $\text{gain}_X(f_X)$ and we put all the points in a range tree data structure D_X [15]. We build an analogous data structure for Y . Next, for each embedding f_Z for Z , we compute the best pair of embeddings (f_X, f_Y) for X and Y as follows.

If there are no interactions between X and Y , we can find the best pair in $\mathcal{O}(\text{polylog } n)$ time by independently picking the best embeddings for X and Y by querying the range trees D_X and D_Y . Indeed, first note that there is no index $i \in [k-1]$ such that $X = \{i, i+1\}$ because

■ **Algorithm 1** Quasi-linear-time algorithm for $k \leq 7$.

```

1: for each feasible irreducible connection  $k$ -pattern  $M$  do
2:   if  $M$  consists of at most two sequential swaps then
3:     Apply the algorithm in Theorem 1.
4:   else
5:     Let  $M = M[X] \uplus M[Y] \uplus M[Z]$  where  $|X| = |Y| = 2$  and  $|Z| = k - 4$ .
6:     if there are no interactions between  $X$  and  $Y$  then
7:       for each embedding  $f_Z$  for  $Z$  do
8:         Independently compute the best embeddings  $f_X$  for  $X$  and  $f_Y$  for  $Y$ .
9:       else
10:        Relax the constraint  $f_X(i) < f_Y(i+1)$  to  $f_X(i) \neq f_Y(i+1)$ .
11:        for each embedding  $f_Z$  for  $Z$  do
12:          Compute the best pair  $(f_X, f_Y)$  satisfying the relaxed constraints.

```

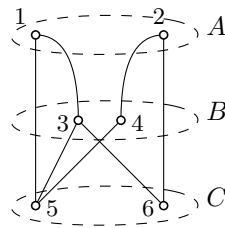
in such a case, both the 2-swap and the remaining $(k-2)$ -swap have to be feasible (similarly for Y). Since there are no interactions between X and Y , we must have $i-1 \in Z \cup \{0\}$ and $i+1 \in Z \cup \{k+1\}$ for every $i \in X \cup Y$. To find the best embedding f_X of $X = \{i, j\}$, we query D_X with the constraints $f_Z(i-1) < f_X(i) < f_Z(i+1)$ and $f_Z(j-1) < f_X(j) < f_Z(j+1)$, where we define $f_Z(0) := 0$ and $f_Z(k+1) := n+1$. We proceed analogously for Y .

Finally, assume there are interactions between X and Y , so from Lemma 6, there is exactly one interaction. W.l.o.g. $i \in X$ and $i+1 \in Y$. Note that $i-1 \in Z \cup \{0\}$ and $i+2 \in Z \cup \{k+1\}$. We first relax the constraint $f_Z(i-1) < f_X(i) < f_Y(i+1) < f_Z(i+2)$, where we define $f_Z(0) := 0$ and $f_Z(k+1) := n+1$, to three constraints $f_Z(i-1) < f_X(i) < f_Z(i+2)$, $f_Z(i-1) < f_Y(i+1) < f_Z(i+2)$, and $f_X(i) \neq f_Y(i+1)$. We then drop the disturbing inequality constraint $f_X(i) \neq f_Y(i+1)$ by color-coding¹. We color each vertex in $[n]$ in red or blue, and we independently pick the best embedding for X (resp. Y) that uses only red (resp. blue) vertices. By using a family of perfect hash functions [5], we can construct a set of $\mathcal{O}(\log^2 n)$ colorings such that, for every pair of embeddings f_X and f_Y , there is at least one coloring that colors all the vertices in f_X red and all the vertices in f_Y blue.

We now obtain the best pair of embeddings (f_X, f_Y) satisfying the relaxed constraints. If the obtained k -swap is not improving, we immediately know that there are no improving k -moves that fit into M . If it is improving and satisfies the original constraint, we are done. Finally, if it is improving but does not satisfy the original constraint, it fits into the connection pattern M' that is obtained from M by swapping i and $i+1$. By Lemma 7, M' is either feasible or reducible. Because no improving k -swaps fit into reducible connection patterns, M' has to be feasible. We therefore obtain a k -move that is as good as the best k -move that fits into M . This completes the proof of Theorem 4.

We finally consider the case of $k = 8$. Note that, because Lemma 6 and 7 do not assume $k \leq 7$, the above algorithm can also compute the best improving k -move that can be decomposed into three sequential swaps of size $(2, 2, k-4)$ for any fixed k under the same assumption. Moreover, any connection patterns of 8-moves consisting of four 2-swaps are reducible because it always induces a pair of two 2-swaps that interact at least twice. The

¹ Instead of color-coding, we can adapt the range tree to support orthogonal range maximum queries with an additional constraint of the form $x \neq i$ by keeping one additional point in each node. With this approach, we can avoid the additional $\log^2 n$ factor. Because this paper does not focus on optimizing the polylog n factor, we do not touch on the details.



■ **Figure 2** An instance of TRIANGLE DETECTION.

remaining case for $k = 8$ is only when the 8-move can be decomposed into three sequential swaps of size $(2, 3, 3)$. In order to tackle this case, we exploit the bounded-weight assumption as follows. For each connection pattern $M = M[X] \uplus M[Y] \uplus M[Z]$ with $|X| = 2$ and $|Y| = |Z| = 3$, and for each embedding f_Z for Z , we want to compute the best pair of embeddings f_X for X and f_Y for Y . When all the weights are integers from $[W]$, the gain of $(f_X, M[X])$ is an integer from $[-2W, 2W]$, and the gain $(f_Y, M[Y])$ is an integer from $[-3W, 3W]$. We therefore have only $\mathcal{O}(W^2)$ pairs of gains. By guessing the pair of gains, the query of finding the *best* pair can be reduced to the query of finding an *arbitrary* pair, and the latter query can be efficiently answered by adapting the range tree. This leads to the following algorithm, whose detailed description is deferred to the full version.

► **Theorem 8.** *When all the weights are integers from $[W]$, there is an $\mathcal{O}(W^2 n \text{ polylog } n)$ -time algorithm to compute the best improving 8-move under the assumption that there are no improving k' -moves for $k' < 8$.*

5 Lower bound for $k = 9$

The starting point for our reduction is the following problem (see Fig. 2 for an exemplary instance).

<p>TRIANGLE DETECTION</p> <p>Input: An undirected graph H whose vertex set $V(H)$ is partitioned into $A \cup B \cup C$.</p> <p>Question: Is there a triple $(a, b, c) \in A \times B \times C$ such that $\{ab, ac, bc\} \subseteq E(H)$?</p>	<p>Parameter: $m := E(H)$.</p>
---	--

We assume without loss of generality that A , B , and C are three independent sets, so that finding such a triple is equivalent to finding a triangle in the graph H . By simple reductions that incur only a constant blow-up in the number of vertices and edges, this problem is equivalent to determining whether a graph has a triangle or not.

► **Assumption 1** (Triangle hypothesis [1]). *There is a fixed $\delta > 0$ such that, in the Word RAM model with words of $\mathcal{O}(\log n)$ bits, any algorithm requires $m^{1+\delta-o(1)}$ time in expectation to detect whether an m -edge graph contains a triangle.*

It should be noted that one can solve TRIANGLE DETECTION in time $\mathcal{O}(n^\omega)$ where n is the number of vertices and $\omega \leq 2.373$ is the best-known exponent for matrix multiplication. Alon et al. [2] found an elegant win-win argument to solve TRIANGLE DETECTION in time $\mathcal{O}(m^{\frac{2\omega}{\omega+1}})$: the 3-vertex paths in which the middle vertex has degree less than $m^{\frac{\omega-1}{\omega+1}}$ can be listed in time $\mathcal{O}(m \cdot m^{\frac{\omega-1}{\omega+1}}) = \mathcal{O}(m^{\frac{2\omega}{\omega+1}})$, and for each, one can check if they form a triangle, whereas the number of vertices of degree greater than $m^{\frac{\omega-1}{\omega+1}}$ is at most $m^{\frac{2}{\omega+1}}$, so one can detect a triangle in time $\mathcal{O}(m^{\frac{2\omega}{\omega+1}})$ in the subgraph that they induce. After more than two decades, this is still the best worst-case running time (when $n^\omega = \Omega(m^{\frac{2\omega}{\omega+1}})$). This suggests

that the triangle hypothesis is likely to hold. Moreover, if one thinks that the above scheme yields the best possible running time and that ω will eventually reach 2, then exponent $4/3$ could be the *right answer* for TRIANGLE DETECTION parameterized by the number of edges. The following is implied by [1, Conjecture 2] (since $\omega \geq 2$), in the regime $m = \Theta(n^{3/2})$ (so that $\mathcal{O}(n^2)$ and $\mathcal{O}(m^{4/3})$ coincide).

► **Assumption 2.** *In the Word RAM model with words of $\mathcal{O}(\log n)$ bits, any algorithm requires $m^{4/3-o(1)}$ time in expectation to detect whether an m -edge $\Theta(m^{2/3})$ -node graph contains a triangle.*

We show that SUBCUBIC 9-OPT DETECTION parameterized by the number of vertices is as hard as TRIANGLE DETECTION parameterized by the number of edges, by providing a linear-time reduction from the latter to the former. In light of Theorem 4, this implies that BOUNDED-DEGREE 8-OPT DETECTION is the only remaining open case where a quasi-linear algorithm is not known but also not ruled out by a standard fine-grained complexity assumption.

► **Lemma 9.** *There is an $\mathcal{O}(m)$ -time reduction from TRIANGLE DETECTION on m -edge graphs to SUBCUBIC 9-OPT DETECTION on $\mathcal{O}(m)$ -vertex undirected graphs with edge weights in $\{1, 2\}$.*

Proof. From a tripartitioned instance of TRIANGLE DETECTION $H = (A \cup B \cup C, E(H))$ with m edges, we build a subcubic graph G with $\Theta(m)$ vertices, an edge-weight function $w : E(G) \rightarrow \{1, 2\}$, and a Hamiltonian cycle \mathcal{C} . From \mathcal{C} , there is a swap of up to 9 edges (i.e., up to 9 deletions and the same number of additions) which results in a lighter Hamiltonian cycle if and only if H has a triangle.

Overall construction of G . We will build G by adding chords to the cycle \mathcal{C} . Henceforth, a *chord* is an edge of G which is not in \mathcal{C} . It is helpful to think of \mathcal{C} as a (subdivided) triangle whose three sides correspond to A , B , and C , which we call the A -side (left), B -side (right), and C -side (bottom), respectively. We will only name the edges of G (and not the vertices), since the problem is more efficiently described in terms of edges. We will define some sequential 3-swaps (we recall that a sequential i -swap is a closed walk of length $2i$ alternating edges of $E(\mathcal{C})$ and edges of $E(G) \setminus E(\mathcal{C})$). Eventually, all the edges that are not in a described sequential 3-swap are incident to a vertex of degree 2, making them undeletable. (One can also enforce that by subdividing every irrelevant edge once.)

The improving 9-move, should there be a triangle abc in H , will consist of a sequence of three 3-swaps. More precisely, it consists of one improving 3-swap, which splits \mathcal{C} into three cycles respectively containing:

- (1) a part of the vertex gadget of some $a \in A$,
- (2) the part of the B -side below the vertex gadget of b , as well as the C -side, and
- (3) the part of the B -side above the vertex gadget of some $b \in N_H(a) \cap B$.

This decreases the total weight by 1. Then a neutral 3-swap reconnects (1) and (2) together, but also detaches (4) a part of the vertex gadget of some $c \in N_H(a) \cap C$. Finally a neutral 3-swap glues (3), (1)+(2), and (4) together, provided $bc \in E(H)$. This results in a new Hamiltonian cycle of length $w(\mathcal{C}) - 1$.

There will be relatively few edges of weight 2. To simplify the presentation, every edge is of weight 1, unless specified otherwise. Let \vec{H} be the directed graph obtained from H by orienting its edges from A to B , from B to C , and from C to A . Note that finding a directed triangle in \vec{H} is equivalent to finding a triangle in H .

Vertex scopes, extended scopes, and nested chords. For $(X, Y) \in \{(A, B), (B, C), (C, A)\}$, we set $Z := \{A, B, C\} \setminus \{X, Y\}$ and we do the following as a preparatory step to encode the arcs of \vec{H} . Each vertex $v \in X$ is given a (pairwise vertex-disjoint) subpath I_v of \mathcal{C} , called the *extended scope* of v , with $|I_v| := 6(|N_H(v) \cap Y|) + 3(|N_H(v) \cap Z|) - 1$ vertices. We think of I_v as being displayed from left to right with the leftmost vertex of index 1, and the rightmost one of index $|I_v|$. The extended scopes of the vertices of A , B , and C occupy respectively the A -side, B -side, and C -side. In what follows, it will be more convenient to have a *circular* notion of *left* and *right*. Starting from the bottom corner of the A -side, and going clockwise to the top corner of the A -side, then down to the bottom corner of the B -side, the relative left and right within the A -side and the B -side coincide with the usual notion as displayed in Figure 3a. But then closing the loop from the right corner of the C -side to its left corner, left and right are switched: the closer to the bottom corner of A (resp. B), the more “right” (resp. “left”).

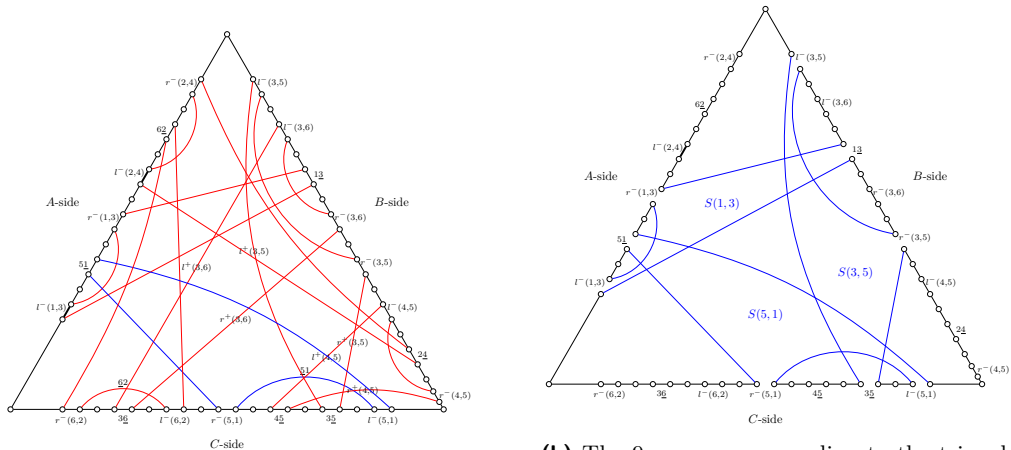
Each vertex $v \in X$ has $|N_H(v) \cap Y|$ nested chords spaced out every three vertices. More precisely, the second vertex of I_v is adjacent to the penultimate, the fifth to the one of index $|I_v| - 4$, the eighth to the one of index $|I_v| - 7$, and so on, until $|N_H(v) \cap Y|$ chords are drawn. Each of these chords is associated to an edge $vy \in E(\{v\}, Y)$, and is denoted by \underline{vy} . A vertex just to the right of the left endpoint, or just to the left of the right endpoint, of such a chord will remain of degree 2 in G . This is the case of the vertices of index 3, 6, ... and $|I_v| - 2, |I_v| - 5, \dots$ in I_v . We call $l^-(v, y)$ (resp. $r^-(v, y)$) the edge of I_v incident to both the left endpoint of \underline{vy} and the vertex just to its left (resp. right endpoint of \underline{vy} and the vertex just to its right). Both endpoints of $l^-(v, y)$ and of $r^-(v, y)$ will eventually have degree 3 in G .

The chord linking the most distant vertices in I_v is called the *outermost* chord, while the one linking the closest pair is called the *innermost* chord. We also say that a chord e is *wider* than a chord e' if e links a farther pair on I_v than e' does. The central path $J_v \subset I_v$ on $|I_v| - (6|N_H(v) \cap Y| - 4) = 3(|N_H(v) \cap Z| + 1)$ vertices, surrounded by the innermost chord, is called the *scope* of v . We map in one-to-one correspondence the edges of $E(\{v\}, Z)$ to every three edges of J_v starting from the third edge (that is, the third, sixth, and so on). Note that we have the exact space to do so, since $|J_v| = 3(|N_H(v) \cap Z| + 1)$. We denote by \underline{zv} the edge in J_v corresponding to the edge $vz \in E(\{v\}, Z)$.

Encoding the arcs of \vec{H} . The last step to encode the arcs of \vec{H} , or equivalently the edges of H , is the following. Keeping the notations of the previous paragraphs, for every edge $xy \in E(X, Y)$, we add two chords (of weight 1): one chord $l^+(x, y)$ between the left endpoint of $l^-(x, y)$ and the right endpoint of \underline{xy} and one chord $r^+(x, y)$ between the right endpoint of $r^-(x, y)$ and the left endpoint of \underline{xy} . We finish the construction of G (and \mathcal{C}) by subdividing each edge between consecutive extended scopes once, to make the resulting edges undeletable. The edges $l^-(a, b)$ for $(a, b) \in A \times B$ get weight 2, while all the other edges of $E(G)$ get weight 1. This finishes the construction of (G, w, \mathcal{C}) . See Figure 3a for an illustration.

Improving and neutral 3-swaps. For each $(x, y) \in E(\vec{H})$, denote by $S(x, y)$ the 3-swap $(\{\underline{xy}, l^-(x, y), r^-(x, y)\}, \{\underline{xy}, l^+(x, y), r^+(x, y)\})$. For $(X, Y) \in \{(A, B), (B, C), (C, A)\}$, we define the set of 3-swaps $S(X, Y) := \bigcup_{xy \in E(X, Y)} S(x, y)$, and $\mathcal{S} := S(A, B) \cup S(B, C) \cup S(C, A)$.

Note that all the 3-swaps of $S(A, B)$ are improving. They gain 1 since $l^-(a, b)$ has weight 2 for any $(a, b) \in A \times B$. On the other hand, all the 3-swaps of $S(B, C)$ and $S(C, A)$ are neutral. The edges added in swaps of \mathcal{S} partition the chords of G , and the open neighborhood



(a) The construction for the instance of Figure 2. Edges of \mathcal{C} are in black, chords are in red, bold edges are the ones with weight 2. The three chords in blue are the edges to add to perform the neutral 3-swap $S(5, 1)$ of $S(\mathcal{C}, A)$.

(b) The 9-move corresponding to the triangle 135 results in a Hamiltonian cycle using one less edge of weight 2. Note that after the swaps $S(1, 3)$ and $S(5, 1)$ are performed, the only 3-swap that can reconnect the three cycles into one, is $S(3, 5)$, implying the existence of the edge 35, and thereby of the triangle 135.

■ **Figure 3** Illustration of the reduction (left) and of a potential solution (right).

of the six vertices involved in every swap are six vertices of degree 2 in G . Therefore, all the possible swaps are in the set \mathcal{S} , they are on vertex-disjoint sets of vertices, and any move is a sequence of 3-swaps of \mathcal{S} .

The vertices of \mathcal{C} are incident to at most one chord. Hence the graph G is subcubic. It has $\sum_{v \in V(H)} 1 + |I_v| \leq 9|E(H)| + |V(H)| = \Theta(m)$ vertices and (G, w, \mathcal{C}) takes $\Theta(m)$ -time to build. To summarize, we defined a linear reduction from TRIANGLE DETECTION with parameter m to SUBCUBIC 9-OPT DETECTION with parameter n . So a quasi-linear algorithm for the latter would yield an unlikely quasi-linear algorithm for the former. We now check that the reduction is correct.

A triangle in H implies an improving 9-move for (G, w, \mathcal{C}) . Let abc be a triangle in H . In particular, all three swaps $S(a, b)$, $S(b, c)$, and $S(c, a)$ exist. Performing these three 3-swaps results in a spanning union of (vertex-disjoint) cycles, whose total weight is $w(\mathcal{C}) - 1$. Indeed $S(a, b)$ is swap of weight -1 , while $S(b, c)$, and $S(c, a)$ are both neutral.

We thus only need to show that the three swaps result in a connected graph (hence, Hamiltonian cycle of lighter weight). By performing the 3-swap $S(a, b)$, we create three components: (1) one on a vertex set $K_{a,b}$ such that $J_a \subseteq K_{a,b} \subseteq I_a$, (2) one containing the scopes of vertices of the B -side to the right (lower part) of the scope of b , and (3) one containing the scopes of vertices of the B -side to the left (upper part) of the scope of b . Then the swap $S(c, a)$ glues (1) and (2) together, but also disconnects (4) a cycle on a vertex set $K_{c,a}$ such that $J_c \subseteq K_{c,a} \subseteq I_c$. At this point, there are three cycles: (3), (1)+(2), and (4). It turns out that the 3-swap $S(b, c)$ deletes exactly one edge in each of these three cycles: b_c in (4), $l^-(b, c)$ in (3), and $r^-(b, c)$ in (1)+(2). Therefore, $S(b, c)$ reconnects these three components into one Hamiltonian cycle.

An improving k -move for (G, w, \mathcal{C}) with $k \leq 9$ implies a triangle in H . We assume that there is an improving k -move $\mathcal{M} = (E^-, E^+)$ for (G, w, \mathcal{C}) with $k \leq 9$. Being improving, the k -move has to contain at least one improving 3-swap of $S(A, B)$. Let $S(a, b)$ be a 3-swap of

$S(A, B)$ in \mathcal{M} such that for every other (improving) 3-swap $S(a, b')$ in \mathcal{M} , the chord ab' is wider than ab . Since $S(a, b)$ exists, it holds in particular that $ab \in E(H)$. Performing $S(a, b)$ results in the union of three cycles: (1) on a vertex set $K_{a,b}$ with $J_a \subseteq K_{a,b} \subseteq I_a$, and cycles (2) and (3) as described in the previous paragraph.

By the choice of b , the only remaining swaps of \mathcal{M} touching $K_{a,b}$ are in $S(C, A)$. So \mathcal{M} has to contain a neutral 3-swap $S(c, a)$ for some $c \in C$. This implies that $ac \in E(H)$. Performing this swap results in three cycles: (3), (1)+(2), and (4), as described above. To reconnect all three components into one Hamiltonian cycle, the 3-swap has to delete exactly one edge in (3), (1)+(2), and (4). The only 3-swap that does so is $S(b, c)$. This finally implies that $bc \in E(H)$. Thus abc is a triangle in H . ◀

We obtain the following theorem as a direct consequence of the previous lemma.

► **Theorem 10.** SUBCUBIC 9-OPT DETECTION *requires time:*

- (1) $n^{1+\delta-o(1)}$ for a fixed $\delta > 0$, under the triangle hypothesis, and
 - (2) $n^{4/3-o(1)}$, under the strong triangle hypothesis,
- in expectation, even in undirected graphs with edge weights in $\{1, 2\}$.

If we use general integral weights and not just $\{1, 2\}$, we can show a stronger lower bound, by reducing from NEGATIVE EDGE-WEIGHTED TRIANGLE. Again, we can assume that the instance is partitioned into three sets A, B, C , and we look for a triangle abc such that $w'(ab) + w'(bc) + w'(ac) < 0$, where w' gives an integral weight to each edge. A truly subcubic (in the number of vertices) algorithm for this problem would imply one for ALL-PAIRS SHORTEST PATHS, which would be considered a major breakthrough. The assumption that such an algorithm is not possible is called the APSP hypothesis.

We only change the above construction in the weight of the edges $l^-(x, y)$. Now each edge $l^-(x, y)$ gets weight $-w'(xy)$. From a NEGATIVE EDGE-WEIGHTED TRIANGLE-instance with n vertices, we obtain an equivalent instance of SUBCUBIC 9-OPT DETECTION with $\mathcal{O}(n^2)$ vertices, in time $\mathcal{O}(n^2)$. So we derive the following.

► **Theorem 11.** SUBCUBIC 9-OPT DETECTION *requires time* $n^{3/2-o(1)}$, *under the APSP hypothesis.*

6 Lower bound for varying k

In this section we describe the main ideas behind the lower bound for k -OPT DETECTION in subcubic graphs for varying k . The details are deferred to the full version due to space restrictions. The overall approach is similar to the lower bound of Guo et al. [6], in that we give a linear-parameter reduction from the k -PARTITIONED SUBGRAPH ISOMORPHISM problem parameterized by the number of edges k . Marx [14] proved that, assuming the Exponential Time Hypothesis, the problem cannot be solved in time $f(k) \cdot n^{o(k/\log k)}$ for any function f .

The instance created in the reduction of Guo et al. [6] may contain vertices of arbitrarily large degrees. To obtain such a reduction to k -OPT DETECTION in subcubic graphs, an essential ingredient is a *choice gadget* with terminal pairs $(x_0, y_0), \dots, (x_\ell, y_\ell)$ which enforces that sufficiently cheap Hamiltonian cycles that enter at x_i , must leave via the corresponding y_i . The gadget can be implemented by suitable weight settings and vertices of degree at most three. This gadget allows us to enforce synchronization properties, which enforce that an improved Hamiltonian cycle first selects which vertices to use in the image of the subgraph isomorphism,

and then selects incident edges for each selected vertex. By carefully coordinating the gadgets, this allows us to implement the hardness proof by an edge selector strategy. It leads to a proof of the following theorem.

► **Theorem 12.** *There is no function f for which k -OPT DETECTION on n -vertex graphs of maximum degree 3 with edge weights in $\{1, 2\}$ can be solved in time $f(k) \cdot n^{o(k/\log k)}$, unless ETH fails.*

We remark that the lower bound also holds for *permissive* local search algorithms which output an improved Hamiltonian cycle of arbitrarily large Hamming distance to the starting cycle \mathcal{C} , if a cheaper cycle exists in the k -OPT neighborhood of \mathcal{C} .

References

- 1 Amir Abboud and Virginia Vassilevska Williams. Popular Conjectures Imply Strong Lower Bounds for Dynamic Problems. In *Proc. 55th FOCS*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.
- 2 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and Counting Given Length Cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.
- 3 Marek Cygan, Lukasz Kowalik, and Arkadiusz Socala. Improving TSP Tours Using Dynamic Programming over Tree Decompositions. In *Proc. 25th ESA*, volume 87 of *LIPICs*, pages 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPICs.ESA.2017.30.
- 4 Mark de Berg, Kevin Buchin, Bart M. P. Jansen, and Gerhard J. Woeginger. Fine-Grained Complexity Analysis of Two Classic TSP Variants. In *Proc. 43rd ICALP*, volume 55 of *LIPICs*, pages 5:1–5:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
- 5 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a Sparse Table with $O(1)$ Worst Case Access Time. *J. ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.
- 6 Jiong Guo, Sepp Hartung, Rolf Niedermeier, and Ondrej Suchý. The Parameterized Complexity of Local Search for TSP, More Refined. *Algorithmica*, 67(1):89–110, 2013. doi:10.1007/s00453-012-9685-8.
- 7 Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Math. Program.*, 1(1):6–25, 1971.
- 8 Keld Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000. doi:10.1016/S0377-2217(99)00284-2.
- 9 Keld Helsgaun. General k -opt submoves for the Lin-Kernighan TSP heuristic. *Math. Program. Comput.*, 1(2-3):119–163, 2009.
- 10 D. S. Johnson and L. A. McGeoch. Experimental Analysis of Heuristics for the STSP. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 369–443. Kluwer Academic Publishers, Dordrecht, 2002.
- 11 D.S. Johnson and L.A McGeoch. The traveling salesman problem: A case study in local optimization. In E. Aarts and J.K. Lenstra, editors, *Local search in combinatorial optimization*, pages 215–310. Wiley, Chichester, 1997.
- 12 S. Lin and Brian W. Kernighan. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research*, 21(2):498–516, 1973. doi:10.1287/opre.21.2.498.
- 13 Dániel Marx. Searching the k -change neighborhood for TSP is $W[1]$ -hard. *Oper. Res. Lett.*, 36(1):31–36, 2008. doi:10.1016/j.orl.2007.02.008.
- 14 Dániel Marx. Can You Beat Treewidth? *Theory of Computing*, 6(1):85–112, 2010. doi:10.4086/toc.2010.v006a005.
- 15 Franco P. Preparata and Michael Ian Shamos. *Computational Geometry - An Introduction*. Texts and Monographs in Computer Science. Springer, 1985.

Linear Transformations Between Colorings in Chordal Graphs

Nicolas Bousquet 

Univ. Grenoble Alpes, CNRS, Grenoble INP, G-SCOP, France
nicolas.bousquet@grenoble-inp.fr

Valentin Bartier

Univ. Grenoble Alpes, CNRS, Grenoble INP, G-SCOP, France
valentin.bartier@grenoble-inp.fr

Abstract

Let k and d be such that $k \geq d + 2$. Consider two k -colorings of a d -degenerate graph G . Can we transform one into the other by recoloring one vertex at each step while maintaining a proper coloring at any step? Cereceda et al. answered that question in the affirmative, and exhibited a recolouring sequence of exponential length.

If $k = d + 2$, we know that there exists graphs for which a quadratic number of recolorings is needed. And when $k = 2d + 2$, there always exists a linear transformation. In this paper, we prove that, as long as $k \geq d + 4$, there exists a transformation of length at most $f(\Delta) \cdot n$ between any pair of k -colorings of chordal graphs (where Δ denotes the maximum degree of the graph). The proof is constructive and provides a linear time algorithm that, given two k -colorings c_1, c_2 computes a linear transformation between c_1 and c_2 .

2012 ACM Subject Classification Mathematics of computing \rightarrow Graph theory

Keywords and phrases graph recoloring, chordal graphs

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.24

Related Version A full version of the paper is available at <http://arxiv.org/abs/1907.01863>.

Funding This work was supported by ANR project GrR (ANR-18-CE40-0032).

1 Introduction

Reconfiguration problems consist in finding step-by-step transformations between two feasible solutions of a problem such that all intermediate states are also feasible. Such problems model dynamic situations where a given solution already in place has to be modified for a more desirable one while maintaining some properties throughout the transformation. Reconfiguration problems have been studied in various fields such as discrete geometry [6], optimization [1] or statistical physics [22] in order to transform, generate, or count solutions. In the last few years, graph reconfiguration received a considerable attention, e.g. reconfiguration of independent sets [3, 21], matchings [7, 20], dominating sets [24] or fixed-degree sequence graphs [9]. For a complete overview of the reconfiguration field, the reader is referred to the two recent surveys on the topic [23, 25].

Two main questions are at the core of combinatorial reconfiguration. (i) Is it possible to transform any solution into any other? (ii) If yes, how many steps are needed to perform this transformation? These two questions and their algorithmic counterparts received considerable attention.



© Nicolas Bousquet and Valentin Bartier;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 24; pp. 24:1–24:15



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Graph recoloring

Throughout the paper, $G = (V, E)$ denotes a graph, $n = |V|$, Δ denotes the maximum degree of G , and k is an integer. For standard definitions and notations on graphs, we refer the reader to [15]. A (proper) k -coloring of G is a function $f : V(G) \rightarrow \{1, \dots, k\}$ such that, for every edge $xy \in E$, we have $f(x) \neq f(y)$. Since we will only consider proper colorings, we will then omit the proper for brevity. The *chromatic number* $\chi(G)$ of a graph G is the smallest k such that G admits a k -coloring. Two k -colorings are *adjacent* if they differ on exactly one vertex. The k -reconfiguration graph of G , denoted by $\mathcal{G}(G, k)$ and defined for any $k \geq \chi(G)$, is the graph whose vertices are k -colorings of G , with the adjacency condition defined above. Cereceda, van den Heuvel and Johnson provided an algorithm to decide whether, given two 3-colorings, one can transform one into the other in polynomial time and characterized graphs for which $\mathcal{G}(G, 3)$ is connected [12, 13]. Given any two k -colorings of a graph, it is **PSPACE**-complete to decide whether one can be transformed into the other for $k \geq 4$ [5].

The k -recoloring diameter of a graph G is the diameter of $\mathcal{G}(G, k)$ if $\mathcal{G}(G, k)$ is connected and is $+\infty$ otherwise. In other words, it is the minimum D for which any k -coloring can be transformed into any other one through a sequence of at most D recolorings. Bonsma and Cereceda [5] proved that there exists a class \mathcal{C} of graphs and an integer k such that, for every graph $G \in \mathcal{C}$, there exist two k -colorings whose distance in the k -reconfiguration graph is finite and super-polynomial in n .

A graph G is d -degenerate if any subgraph of G admits a vertex of degree at most d . In other words, there exists an ordering v_1, \dots, v_n of the vertices such that for every i , v_i has at most d neighbors in v_{i+1}, \dots, v_n . It was shown independently by Dyer et al [16] and by Cereceda et al. [12] that for any d -degenerate graph G and every $k \geq d + 2$, $\mathcal{G}(G, k)$ is connected. Note that the bound on k is the best possible since the $\mathcal{G}(K_n, n)$ is not connected. Cereceda [11] conjectured the following:

► **Conjecture 1** (Cereceda [11]). *For every d , every $k \geq d + 2$, and every d -degenerate graph G , the diameter of $\mathcal{G}(G, k)$ is at most $C_d \cdot n^2$.*

If true, the quadratic function is the best possible, even for paths, as shown in [2]. Bousquet and Heinrich [8] recently proved that the diameter of $\mathcal{G}(G, k)$ is $O(n^{d+1})$. In the general case, Cereceda's conjecture is only known to be true for $d = 1$ (trees) [2] and $d = 2$ and $\Delta \leq 3$ [18]. The diameter of $\mathcal{G}(G, k)$ is $O(n^2)$ when $k \geq \frac{3}{2}(d + 1)$ as shown in [8]. Even if Cereceda's conjecture is widely open for general graphs, it has been proved for a few graph classes, e.g. chordal graphs [2], bounded treewidth graphs [4], and bipartite planar graphs [8].

Jerrum conjectured that if $k \geq \Delta + 2$, the mixing time (time needed to approach the stationary distribution) of the Markov chain of graph colorings¹ is $O(n \log n)$. So far, the conjecture has only been proved if $k \geq (\frac{11}{6} - \epsilon)\Delta$ [14]. Since the diameter of the reconfiguration graph is a lower bound of the mixing time, a lower bound on the diameter is of interest to study the mixing time of the Markov chain. In order to obtain such a mixing time, we need an (almost) linear diameter.

The diameter of $\mathcal{G}(G, k)$ is linear if $k \geq 2d + 2$ [10] or if k is at least the Grundy number of G plus 1 [4]. When $k = d + 2$, the diameter of $\mathcal{G}(G, k)$ may be quadratic, even for paths [2]. But it might be true that the diameter of $\mathcal{G}(G, k)$ is linear whenever $k \geq d + 3$. In this paper, we investigate the following question, raised for instance in [8]: when does the k -recoloring diameter of d -degenerate graphs become linear?

¹ A random walk on $\mathcal{G}(G, k)$. For more details on Markov chains on graph colorings, the reader is for instance referred to [14].

Our results

A graph is *chordal* if it does not contain any induced cycle of length at least 4. Chordal graphs admit a *perfect elimination ordering*, i.e. there exists an ordering v_1, \dots, v_n of V such that, for every i , $N[v_i] \cap \{v_{i+1}, \dots, v_n\}$ is a clique. Chordal graphs are $(\omega(G) - 1)$ -degenerate where $\omega(G)$ is the size of a maximum clique of G . Our main result is the following:

► **Theorem 2.** *Let Δ be a fixed integer. Let G be a d -degenerate chordal graph of maximum degree Δ . For every $k \geq d+4$, the diameter of $\mathcal{G}(G, k)$ is at most $O_\Delta(n)$. Moreover, given two colorings c_1, c_2 of G , a transformation of length at most $O_\Delta(n)$ can be found in linear time.*

Theorem 2 improves the best existing upper bound on the diameter of $\mathcal{G}(G, k)$ (where G is chordal) which was quadratic up to $k = 2d + 1$ [10].

Note that the bound on k is almost the best possible since we know that this result cannot hold for $k \leq d + 2$ [2]. So there is only one remaining case which is the case $k = d + 3$.

► **Question 3.** *Is the diameter of $\mathcal{G}(G, d + 3)$ at most $f(\Delta(G)) \cdot n$ for any d -degenerate graph G ?*

In some very restricted cases (such as power of paths), our proof technique can be extended to $k = d + 3$, but this is mainly due to the very strong structure of these graphs. For chordal graphs (or even interval graphs), we need at least $d + 4$ colors at several steps of the proof and decreasing k to $d + 3$ seems to be a challenging problem.

We also ask the following question: is it possible to remove the dependency on Δ to only obtain a dependency on the degeneracy? More formally:

► **Question 4.** *Is the diameter of $\mathcal{G}(G, d + 3)$ at most $f(d) \cdot n$ for any d -degenerate chordal graph G ?*

The best known result related to that question is the following: $\mathcal{G}(G, k)$ has linear diameter if $k \geq 2d + 2$ (and f is a constant function) [10].

► **Question 5.** *Is the diameter of $\mathcal{G}(G, d + 3)$ at most $f(\Delta(G)) \cdot n$ for any bounded treewidth graph G ?*

Our proof cannot be immediately adapted for bounded treewidth graphs since we use the fact that all the vertices in a bag have distinct colors. Feghali [17] proposed a method to “chordalize” bounded treewidth graphs for recoloring problems. However his proof technique does not work here since it may increase the maximum degree of the graph. We nevertheless think that our proof technique can be adapted in order to study many well-structured graph classes.

Proof outline

In order to prove Theorem 2, we introduce a new proof technique to obtain linear diameters for recoloring graphs. The existing results (e.g. [10]) ensuring that $\mathcal{G}(G, k)$ has linear diameter are based on inductive proofs that completely fail when k is close to d . On the other hand, in the proofs giving quadratic diameters (e.g. [2]), the technique usually consists in finding two vertices that can be “identified” and then applying induction on the reduced graph. In that case, even if we can identify two vertices after a constant number of single vertex recolorings, we only obtain a quadratic diameter (since each vertex might “represent” a linear number of initial vertices). Both approaches are difficult to adapt to obtain linear transformations since they do not use or “forget” the original structure of the graph.

Let us roughly describe the idea of our method on interval graphs. A buffer \mathcal{B} is a set of vertices contained in $f(\omega)$ consecutive cliques of the clique path. We assume that at the left of the buffer, the coloring of the graph already matches the target coloring. We moreover assume that the coloring of \mathcal{B} is special in the sense that, for every vertex v in \mathcal{B} , at most $d + 2$ colors appear in the neighborhood of v ². Note that in order to satisfy this property (and several others detailed in Section 2), the buffer has to be “long enough”. The main technical part of the proof consists in showing that if the buffer is “long enough”, then we can modify the colors of vertices of the buffer in such a way that the same assumptions hold for the buffer starting one clique to the right of the starting clique of \mathcal{B} . We simply have to repeat at most n times this operation to guarantee that the coloring of the whole graph is the target coloring. Since a vertex is recolored only if it is in the buffer and a vertex is in the buffer a constant (assuming Δ constant) number of times, every vertex is recolored a constant number of times.

The structure of this special coloring of the buffer, which is the main novelty of this paper, is described in Sections 2.2 to 2.4. We actually show that this graph recoloring problem can be rephrased into a “vectorial recoloring problem” (Section 2.5) which is easier to manipulate. And we finally prove that this vectorial recoloring problem can be solved by recoloring every element (and then every vertex of the graph) at most a constant number of times in Section 3.

2 Buffer and vectorial coloring

Throughout this section, $G = (V, E)$ is a chordal graph on n vertices of maximum clique number ω and maximum degree Δ . Let $k \geq \omega + 3$ be the number of colors denoted by $1, \dots, k$. Given two integers $x \leq y$, $\llbracket x, y \rrbracket$ is the set $\{x, x + 1, \dots, y\}$. The *closed neighbourhood* of a set $S \subseteq V$ is $N[S] := S \cup (\cup_{v \in S} N(v))$.

2.1 Chordal graphs and clique trees

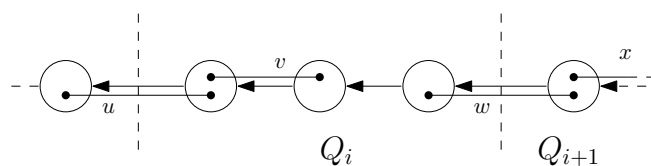
Vertex ordering and canonical coloring

Let v_1, v_2, \dots, v_n be a perfect elimination ordering of V . A greedy coloring of v_n, v_{n-1}, \dots, v_1 gives an optimal coloring c_0 of G using only ω colors. The coloring c_0 is called the *canonical coloring of G* . The colors $c \in 1, 2, \dots, \omega$ are called the *canonical colors* and the colors $c > \omega$ are called the *non-canonical colors*. Note that the independent sets $X_i := \{v \in V \text{ such that } c_0(v) = i\}$ for $i \leq \omega$, called the *classes* of G , partition the vertex set V .

Clique tree

Let $G = (V, E)$ be a chordal graph. A *clique tree* of G is a pair (T, B) where $T = (W, E')$ is a tree and B is a function that associates to each node U of T a subset of vertices B_U of V (called *bag*) such that: (i) every bag induces a clique, (ii) for every $x \in V$, the subset of nodes whose bags contain x forms a non-empty subtree in T , and (iii) for every edge $(U, W) \in T$, $B_U \setminus B_W$ and $B_W \setminus B_U$ are non empty. Note that the size of every bag is at most $\omega(G)$. A clique-tree of G can be found in linear time [19]. Throughout this section, $T = (V_T, E_T)$ is a clique tree of G . We assume that T is rooted on an arbitrary node. Given a rooted tree T and a node C of T , the *height* of C denoted by $h(C)$ is the length of the path from the root to C .

² Our condition is actually even more restrictive.



■ **Figure 1** The nodes represent cliques of G . The vertices v and w belong to Q_i . The vertex u does not belong to Q_i even if it intersects cliques of Q_i , and the vertex x belongs to Q_{i+1} .

► **Observation 6.** Let G be a chordal graph of maximum degree Δ and T be a clique tree of G rooted in an arbitrary node. Let x be a vertex of G and C_i, C_j be two bags of T that contain x . Then $h(C_j) - h(C_i) \leq \Delta$.

Proof. We can assume without loss of generality (free to replace the one with the smallest height by the first common ancestor of C_i and C_j) that C_i is an ancestor of C_j (indeed, this operation can only increase the difference of height). Let P be the path of T between C_i and C_j and (U, W) be an edge of P with $h(U) < h(W)$. By assumption on the clique tree, there is a vertex y that appears in B_W and that does not appear in B_U . Since this property is true for every edge of T and since all the bags of P induce cliques and contain x , the vertex x has at least $|P|$ neighbors. ◀

2.2 Buffer, blocks and regions

Let C_e be a clique of T . We denote by T_{C_e} the subtree of T rooted in C_e and by $h_{C_e}(C)$ the height of the clique $C \in T_{C_e}$. Given a vertex $v \in T_{C_e}$, we say that v starts at height h if the maximum height of a clique of T_{C_e} containing v is h (in T_{C_e}).

Let $s := 3\binom{\omega}{2} + 2$ and $N = s + k - \omega + 1$ where k is the number of colors. The *buffer* \mathcal{B} rooted in C_e is the set of vertices of G that start at height at most $3\Delta N - 1$ in T_{C_e} . For every $0 \leq i \leq 3N - 1$, the *block* Q_{3N-i} of \mathcal{B} is the set of vertices of G that start at height h with $i\Delta \leq h \leq (i+1)\Delta - 1$. Finally, for $0 \leq i \leq N - 1$, the *region* R_i of \mathcal{B} is the set of blocks $Q_{3i+1}, Q_{3i+2}, Q_{3(i+1)}$. Unless stated otherwise, we will always denote the three blocks of R_i by A_i, B_i and C_i , and the regions of a buffer \mathcal{B} by R_1, \dots, R_N . Given a color class X_p and $S \subseteq V$, we denote by $N[S, p]$ the set $N[S \cap X_p]$. By definition of a block and Observation 6 we have:

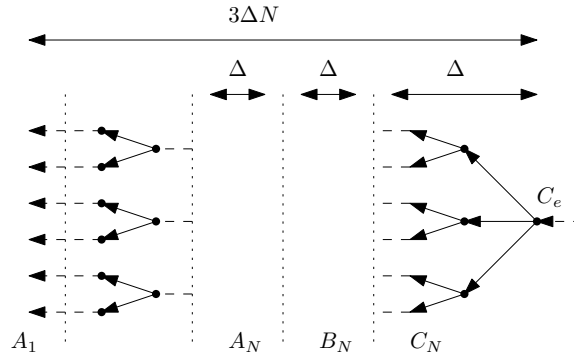
► **Observation 7.** Let C_e be a clique of T and \mathcal{B} be the buffer rooted in T_{C_e} . Let Q_{i-1}, Q_i, Q_{i+1} be three consecutive blocks of \mathcal{B} . Then $N[Q_i] \subseteq Q_{i-1} \cup Q_i \cup Q_{i+1}$. In particular for each region $R_i = (A_i, B_i, C_i)$ of \mathcal{B} , $N[B_i] \subseteq R_i$.

Proof. Let v be a vertex of Q_i . By definition of Q_i , v starts at height h with $(3N - i)\Delta \leq h \leq (3N - i + 1)\Delta - 1$. Let u be a neighbour of v . By Observation 6, u starts at height h' with $h - \Delta \leq h' \leq h + \Delta$, thus we have $(3N - i - 1)\Delta \leq h' \leq (3N - i + 2)\Delta - 1$. It follows that u belongs either to Q_{i-1}, Q_i , or Q_{i+1} . ◀

We refer to this property as the *separation property*. It implies that when recoloring a vertex of Q_i , one only has to show that the coloring induced on Q_{i-1}, Q_i, Q_{i+1} remains proper.

2.3 Vectorial coloring

Let \mathcal{B} be a buffer. We denote the set of vertices of class p that belong to the sequence of blocks Q_i, \dots, Q_j of \mathcal{B} by (Q_i, \dots, Q_j, p) . A *color vector* ν is a vector of size ω such that $\nu(p) \in \llbracket 1, k \rrbracket$ for every $p \in \llbracket 1, \omega \rrbracket$, and $\nu(p) \neq \nu(q)$ for every $p \neq q \leq \omega$. A block Q



■ **Figure 2** The buffer \mathcal{B} rooted at C_e . The dots represent cliques of T and the dashed lines separates the blocks of \mathcal{B} .

is *well-colored* for a color vector ν_Q if all the vertices of (Q, p) are colored with $\nu_Q(p)$. It does not imply that all the colors are $\leq \omega$ but just that all the vertices of a same class have the same color (and vertices of different classes have different colors). For brevity, we say that (Q, ν_Q) is *well-colored* and when ν_Q is clear from context we just say that Q is *well-colored*. In particular, a well-colored block is properly colored. Since the set (Q, p) may be empty, a block may be well-colored for different vectors. However, a color vector defines a unique coloring of the vertices of a block (if (Q, ν_Q) is well-colored then every vertex of (Q, p) has to be colored with $\nu_Q(p)$). The color vector ν is *canonical* if $\nu(p) = p$ for every $p \leq \omega$. A sequence of blocks Q_1, \dots, Q_r is *well-colored* for (ν_1, \dots, ν_r) if (Q_i, ν_i) is well-colored for every $i \leq r$.

► **Definition 8** (Waiting region). A region R well-colored for vectors ν_A, ν_B, ν_C is a *waiting region* if $\nu_A = \nu_B = \nu_C$.

► **Definition 9** (Color region). A region R well-colored for vectors ν_A, ν_B, ν_C is a *color region* if there exist a canonical color c_1 , a non-canonical color z and a class p such that:

1. $\nu_A(m) = \nu_B(m) = \nu_C(m) \notin \{c_1, z\}$ for every $m \neq p$.
2. $\nu_A(p) = c_1$ and $\nu_B(p) = \nu_C(p) = z$.

In other words, the color of exactly one class is modified from a canonical color to a non-canonical color between blocks A and C . We say that the color c_1 *disappears* in R and that the color z *appears* in R . For brevity we say that R is a color region for the class X_p and colors c_1, z .

► **Definition 10** (Transposition region). A region R well-colored for vectors ν_A, ν_B, ν_C is a *transposition region* if there exist two canonical colors $c_1 \neq c_2$, two non-canonical colors $z \neq z'$ and two distinct classes p, q such that:

1. $\nu_A(m) = \nu_B(m) = \nu_C(m) \notin \{c_1, c_2, z, z'\}$ and is canonical for every $m \notin \{p, q\}$.
2. $\nu_A(p) = c_1, \nu_B(p) = z, \nu_C(p) = c_2$.
3. $\nu_A(q) = c_2, \nu_B(q) = z', \nu_C(q) = c_1$.

Note that ν_A and ν_C only differ on the coordinates p and q which have been permuted. The colors z and z' are called the *temporary colors* of R . Note that the coloring induced on R is proper since the separation property ensures that $N[A \cap R] \subseteq A \cup B$, $N[C \cap R] \subseteq B \cap C$ and no class in B is colored with c_1 nor c_2 .

Let ν be a color vector. The color vector ν' is obtained from ν by *swapping the coordinates* $p, \ell \leq \omega$ if for every $m \notin \{p, \ell\}$, $\nu'(m) = \nu(m)$, $\nu'(p) = \nu(\ell)$, and $\nu'(\ell) = \nu(p)$. In other words, ν' is the vector obtained from ν by permuting the coordinates p and ℓ .

	A_i	B_i	C_i		A_i	B_i	C_i		A_i	B_i	C_i
X_1	1	1	1		1	1	1		1	<u>5</u>	3
X_2	3	3	3		3	5	5		3	<u>6</u>	1
X_3	4	4	4		4	4	4		4	4	4
X_4	2	2	2		2	2	2		2	2	2
	Waiting region				Color region				Transposition region		

■ **Figure 3** Example of waiting, color, and transposition regions with $\omega = 4$. Each square represents a region, the dotted lines separate the blocks and the dashed lines separate the classes. The colors 1 to 4 are the canonical colors and the colors 5 and 6 are non-canonical colors. The underlined colors in the transposition region indicate the temporary colors.

Swapping the coordinates p and ℓ in a region R well-colored for (ν_A, ν_B, ν_C) means that for every block $Q \in \{A, B, C\}$, we replace ν_Q by the color vector ν'_Q obtained by swapping the coordinates p and ℓ of ν_Q . It does not refer to a reconfiguration operation but just to an operation on the vectors.

► **Observation 11.** *Swapping two coordinates in a waiting (resp. color, resp. transposition) region leaves a waiting (resp. color, resp. transposition) region.*

Using the following lemma, we can assume that all the transposition regions use the same temporary colors.

► **Lemma 12.** *Let R be a transposition region with temporary colors z, z' . Let $z'' \notin \{z, z'\}$ be a non-canonical color. By recoloring the vertices of R at most once, we can assume the temporary colors are z, z'' .*

Proof. Let p and q be the coordinates which are permuted in R . By definition of transposition regions, no vertex of R is colored with z'' . As any class is an independent set and by the separation property, we can recolor (B, q) with z'' to obtain the desired coloring of R . ◀

2.4 Valid and almost valid buffers

In what follows, a bold symbol ν always denote a tuple of vectors and a normal symbol ν always denotes a vector. Let $\mathcal{B} = R_1, R_2, \dots, R_N$ be a buffer such that all the regions $R_i = A_i, B_i, C_i$ are well-colored for the vectors $\nu_{A_i}, \nu_{B_i}, \nu_{C_i}$. So \mathcal{B} is well-colored for $\nu = (\nu_{A_1}, \nu_{B_1}, \nu_{C_1}, \nu_{A_2}, \dots, \nu_{C_N})$. The buffer (\mathcal{B}, ν) is *valid* if:

1. [Continuity property] For every $i \in 1, 2, \dots, N - 1$, $\nu_{C_i} = \nu_{A_{i+1}}$.
2. The vectors ν_{A_1}, ν_{B_1} and ν_{C_1} are canonical (and then R_1 is a waiting region).
3. The regions R_2, \dots, R_{s-1} define a *transposition buffer*, that is a sequence of consecutive regions that are either waiting regions or transposition regions using the same temporary colors.
4. The regions R_{s+1}, \dots, R_{N-1} define a *color buffer*, that is a sequence of consecutive regions that are either color regions or waiting regions.
5. The regions R_s and R_N are waiting regions.

Note that Property 1 along with the definition of well-colored regions enforce “continuity” in the coloring of the buffer: the coloring of the last block of R_i and the first block of R_{i+1} in a valid buffer have to be the same.

An *almost valid buffer* (\mathcal{B}, ν) is a buffer that satisfies Properties 1 to 4 of a valid buffer and for which Property 5 is relaxed as follows:

5'. The region R_s is a transposition region or a waiting region. R_N is a waiting region.

Let us make a few observations.

► **Observation 13.** *Let (\mathcal{B}, ν) be an almost valid buffer. For every $i \leq s$, the color vectors ν_{A_i} and ν_{C_i} are permutations of the canonical colors.*

Proof. By induction on i . By Property 2 of almost valid buffers, it is true for every $i \leq r$. Suppose now that the property is verified for R_i with $2 < i < s$. By assumption ν_{C_i} is a permutation of $\llbracket 1, \omega \rrbracket$ and the continuity property (Property 1 of almost valid buffer) ensures that $\nu_{A_{i+1}} = \nu_{C_i}$. By Property 3 we only have two cases to consider, either R_{i+1} is a waiting region and by definition $\nu_{C_{i+1}} = \nu_{A_{i+1}}$, or R_{i+1} is a transposition region. In the later case, by definition of a transposition region, $\nu_{C_{i+1}}$ is equal to $\nu_{A_{i+1}}$ up to a transposition of some classes k, ℓ and thus is a permutation of the canonical colors. ◀

► **Observation 14.** *Let (\mathcal{B}, ν) be an almost valid buffer and c be a non-canonical color. There exists a unique class $p \leq \omega$ such that $\nu_{C_s}(p) = c$. Furthermore, either the class p is colored with c on all the blocks of R_{s+1}, \dots, R_N , or the color c disappears in a color region for the class p .*

Proof. By Observation 13, ν_{C_s} is a permutation of the canonical colors. Thus there exists a unique class $p \leq \omega$ such that $\nu_{C_s}(p) = c$. Furthermore, by Property 4 of almost valid buffer, the regions R_{s+1}, \dots, R_N are either waiting or color regions. The continuity property then ensures that either the class p is colored with c on R_{s+1}, \dots, R_N or that the color c disappears in a color region if there exists a color region for the class p . ◀

Since only non-canonical colors can appear in a color region, we have the following observation:

► **Observation 15.** *Let (\mathcal{B}, ν) be an almost valid buffer and z be a non-canonical color. Either no vertex of R_{s+1}, \dots, R_N is colored with z , or there exists a color region R_i with $s < i < N$ for the class p in which z appears. In the latter case, the vertices of the color buffer of \mathcal{B} colored with z are exactly the vertices of $(B_i, C_i, \dots, C_N, p)$.*

Finally, since the number of regions in the color buffer is the number of non-canonical colors, we have:

► **Observation 16.** *Let (\mathcal{B}, ν) be an almost valid buffer. There exists a waiting region in R_{s+1}, \dots, R_{N-1} if and only if there exists a non-canonical color that does not appear in R_{s+1}, \dots, R_{N-1} .*

2.5 Vectorial recoloring

Let (\mathcal{B}, ν) be a buffer. The tuple of color vectors $\nu = (\nu_{Q_1}, \dots, \nu_{Q_{3\Delta N}})$ is a (proper) *vectorial coloring* of \mathcal{B} if for every color c and every $i \leq 3\Delta N - 1$ such that c is in both ν_{Q_i} and $\nu_{Q_{i+1}}$, then there exists a unique class $p \leq \omega$ such that $\nu_{Q_i}(p) = \nu_{Q_{i+1}}(p) = c$.

► **Observation 17.** *Any proper vectorial coloring (\mathcal{B}, ν) induces a proper coloring of $G[\mathcal{B}]$.*

Proof. Indeed, two different classes in two consecutive blocks cannot have the same color in a proper vectorial coloring. Since for any block Q_i of \mathcal{B} and for any class p , $N[Q_i, p] \subseteq Q_{i-1} \cup Q_i \cup Q_{i+1}$, the coloring induced on $G[\mathcal{B}]$ is proper. ◀

Note that if (\mathcal{B}, ν) is an almost valid buffer then ν is a proper vectorial coloring of \mathcal{B} by the continuity property and the definition of waiting, color, and transposition regions. Since we will only consider proper vectorial colorings, we will omit the term proper for brevity.

Let ν_Q be a color vector. A color vector ν'_Q is *adjacent to* ν_Q if there exist a class p and a color $c \notin \nu_Q$ such that $\nu'_Q(p) = c$ and $\nu'_Q(m) = \nu_Q(m)$ for every $m \neq p$.

► **Observation 18.** *Let Q be a block well-colored for ν_Q and let ν'_Q be a color vector adjacent to ν_Q such that $\nu'_Q(p) = c \neq \nu_Q(p)$. Then recoloring the vertices of (Q, p) one by one is a proper sequence of recolorings of $G[Q]$ after which Q is well-colored for ν'_Q .*

Let (ν, ν') be two vectorial colorings of a buffer \mathcal{B} . The coloring ν' is a *vectorial recoloring* of ν if there exists a unique $i \in \llbracket 1, 3\Delta N \rrbracket$ such that ν'_{Q_i} is adjacent to ν_{Q_i} and $\nu'_{Q_j} = \nu_{Q_j}$ for $j \neq i$. By Observation 18, we have:

► **Observation 19.** *Let $t \geq 1$ and (ν^1, ν^t) be two (proper) vectorial colorings of a buffer \mathcal{B} . If there exists a sequence of adjacent (proper) vectorial recolorings $\nu^1, \nu^2, \dots, \nu^t$, then there exists a sequence of (proper) single vertex recolorings of $G[\mathcal{B}]$ after which the coloring of \mathcal{B} is well-colored for ν^t .*

Given a sequence of vectorial recolorings $\nu^1, \nu^2, \dots, \nu^t$, we say that each coordinate is recolored at most ℓ times if for every coordinate $p \leq \omega$ and every $r \in \llbracket 1, 3\Delta N \rrbracket$, there exist at most ℓ indices t_1, \dots, t_ℓ such that the unique difference between ν^{t_i} and $\nu^{t_{i+1}}$ is the p -th coordinate of the r -th vector of the tuples.

3 Algorithm outline

Let G be a chordal graph of maximum degree Δ and maximum clique size ω , T be a clique tree of G , and ϕ be any k -coloring of G . We propose an iterative algorithm that recolors the vertices of the bags of T from the leaves to the root until we obtain the canonical coloring defined in Section 2.1. Let S be a clique of T . A coloring α of G is *treated up to* S if:

1. Vertices starting at height more than $3\Delta N$ in T_S are colored canonically, and
2. The buffer rooted at S is valid.

Let C be a clique of T . We associate a vector ν_C of length ω to the clique C as follows. We set $\nu_C(\ell) = \alpha(v)$ if there exists $v \in X_\ell \cap C$. Then we arbitrarily complete ν_C in such a way all the coordinates of ν_C are distinct (which is possible since $|\nu_C| < k$).

Given two vectors ν and ν' the *difference* $D(\nu, \nu')$ between ν and ν' is $|\{p : \nu(p) \neq \nu'(p)\}|$, i.e. the number of coordinates on which ν and ν' differ. Given an almost valid buffer (\mathcal{B}, ν) and a vector ν_C , the *border error* $D_{\mathcal{B}}(\nu_C, \nu)$ is $D(\nu_{C_N}, \nu_C)$.

Let \mathcal{B} be a buffer. The class $p \leq \omega$ is *internal* to \mathcal{B} if $N[R_N, p] \subseteq R_{N-1} \cup R_N$.

We first state the main technical lemmas of the paper with their proof outlines and finally explain how we can use them to derive Theorem 2. The complete proofs of the lemmas annotated with (*) are included in the full version of the article (see related version).

► **Lemma 20 (*)**. *Let C be a clique associated with ν_C . Let S be a child of C , \mathcal{B} be the buffer rooted at S and ν be a tuple of vectors such that (\mathcal{B}, ν) is valid. If $D_{\mathcal{B}}(\nu_C, \nu) > 0$, then there exists a recoloring sequence of $\cup_{i=s}^N R_i$ such that the resulting coloring ν' satisfies $D_{\mathcal{B}}(\nu_C, \nu') < D_{\mathcal{B}}(\nu_C, \nu)$, and (\mathcal{B}, ν') is almost valid. Moreover, every coordinate of $\cup_{i=s}^N R_i$ is recolored at most 3 times and only internal classes are recolored.*

Outline of the proof. Let ℓ be a class on which ν_C and ν_{C_N} are distinct. Then, in particular, no vertex of X_ℓ is in $C_N \cap C$ thus the class ℓ is internal. Given an internal class ℓ , if we modify $\nu_{C_N}(\ell)$ and maintain a proper vectorial coloring of the buffer \mathcal{B} , then the corresponding recoloring of the graph is proper. So, if we only recolor internal classes of R_N , then we simply have to check that the vectorial coloring of \mathcal{B} remains proper. The proof is then based on a case study depending on whether $\nu_C(\ell)$ is canonical or not. A more complete sketch is given in Section 3.1 ◀

► **Lemma 21 (*)**. *Let (\mathcal{B}, ν) be an almost valid buffer. There exists a recoloring sequence of $\cup_{i=2}^s R_i$ such that every coordinate is recolored at most 6 times and the resulting coloring ν' is such that (\mathcal{B}, ν') is valid.*

Outline of the proof. The proof distinguishes two cases: either there exists a waiting region in the transposition buffer or not. In the first case, we show that we can “slide” the waiting regions to the right of the transposition buffer and then ensure that R_s is a waiting region. Otherwise, because of the size of the transposition buffer, then some pair of colors has to be permuted twice. In this case, we show that these two transpositions can be replaced by waiting regions (and we can apply the first case). A more complete sketch is given in Section 4. ◀

Note that given a clique C and its associated vector ν_C , applying Lemma 21 to an almost valid buffer (\mathcal{B}, ν) rooted at a child S of C does not modify $D_{\mathcal{B}}(\nu_C, \nu)$ since the region R_N is not recolored.

Let C be a clique and S_1, S_2 be two children of C . For every $i \leq 2$, let \mathcal{B}_i be the buffer of S_i and assume that \mathcal{B}_i is valid for ν^i . We say that \mathcal{B}_1 and \mathcal{B}_2 have *the same coloring* if $\nu^1 = \nu^2$.

► **Lemma 22 (*)**. *Let C be a clique associated with ν_C . Let S_1, S_2, \dots, S_e be the children of C , and for every $i \leq e$, \mathcal{B}_i be the buffer rooted at S_i . Let ν^i be a vectorial coloring such that (\mathcal{B}_i, ν^i) is valid. If $D_{\mathcal{B}_i}(\nu_C, \nu^i) = 0$ for every $i \leq e$, then there exists a recoloring sequence of $\cup_{j=2}^{N-1} R_j^i$ such that every coordinate is recolored $O(\omega^2)$ times, the final coloring of all the \mathcal{B}_i s is the same coloring ν' , $D_{\mathcal{B}_i}(\nu_C, \nu') = 0$, and (\mathcal{B}_i, ν') is valid for every $i \leq e$.*

Outline of the proof. First, we prove that it is possible to transform the coloring of \mathcal{B}_i in such a way that all the color buffers have the same coloring, and that $\nu_s^1 = \nu_s^i$ for $i \in [2, e]$. We then have to ensure that the vectors of the transposition buffers are the same, which is more complicated. Indeed, even if we know that the vectors ν_s^1 and ν_s^i are the same, we are not sure that we use the same sequence of transpositions in the transposition buffers of \mathcal{B}_1 and \mathcal{B}_i to obtain it. Let τ_1, \dots, τ_r be the set of transpositions of \mathcal{B}_1 . The proof consists in showing that we can add to \mathcal{B}_i the transpositions $\tau_1, \dots, \tau_r, \tau_r^{-1}, \dots, \tau_1^{-1}$ at the beginning of the transposition buffer. It does not modify ν_{A_s} since this sequence of transpositions gives the identity. Finally, we prove that $\tau_r^{-1}, \dots, \tau_1^{-1}$ can be cancelled with the already existing transpositions of \mathcal{B}_i . And then the transposition buffer of \mathcal{B}_i only consists of τ_1, \dots, τ_r . ◀

► **Lemma 23 (*)**. *Let C be a clique of T with children S_1, S_2, \dots, S_e and let α be a k -coloring of G treated up to S_i for every $i \in [1, e]$. Let ν_C be a vector associated with C and $\mathcal{B}_i = R_1^i, \dots, R_N^i$ denote the buffer rooted at S_i . Assume that there exists ν such that (\mathcal{B}_i, ν) is valid and satisfies $D_{\mathcal{B}_i}(\nu_C, \nu) = 0$ for every $i \leq e$. Then there exists a recoloring sequence of $\cup_{j=2}^{N-1} R_j^i$ such that, for every $i \leq e$, every vertex of \mathcal{B}_i is recolored at most one time and such that the resulting coloring of G is treated up to C .*

Outline of the proof. This proof “only” consists in shifting the buffer of one level. We simply recolor the vertices that now start in another region (of the buffer rooted at C) with their new color. We prove that the recoloring algorithm cannot create any conflict. ◀

Given Lemmas 20, 21, 22 and 23 we can prove our main result:

► **Theorem 24.** *Let Δ be a fixed constant. Let $G(V, E)$ be a d -degenerate chordal graph of maximum degree Δ and ϕ be any k -coloring of G with $k \geq d + 4$. Then we can recolor ϕ into the canonical coloring c_0 in at most $O(d^4 \Delta \cdot n)$ steps. Moreover the recoloring algorithm runs in linear time.*

Proof. Let c_0 be the canonical coloring of G as defined in Section 2.1, and T be a clique tree of G . Let us first show that given a clique $C \in T$ with children S_1, \dots, S_e and a coloring α treated up to S_i for every $i \leq e$, we can obtain a coloring of G treated up to C . Let ν_C be a vector associated with C . For every $i \leq e$, let \mathcal{B}_i be the buffer rooted in S_i and ν^i be a vectorial coloring of \mathcal{B}_i such that (\mathcal{B}_i, ν^i) is valid. For every $i \leq e$, by applying Lemmas 20 and 21 at most $D_{\mathcal{B}_i}(\nu_C, \nu^i)$ times to (\mathcal{B}_i, ν^i) , we obtain a vectorial coloring ν^i such that (\mathcal{B}_i, ν^i) is valid and $D_{\mathcal{B}_i}(\nu_C, \nu^i) = 0$. By Lemma 22, we can recolor each ν^i into ν' such that for every i , (\mathcal{B}_i, ν') is valid and $D_{\mathcal{B}_i}(\nu_C, \nu') = 0$. Then we can apply Lemma 23 to obtain a coloring of G such that the buffer (\mathcal{B}, ν) rooted in C is valid. Since no vertex starting in cliques $W \in T_C$ with $h_C(W) > 3\Delta N$ is recolored, these vertices remain canonically colored and the resulting coloring of G is treated up to C . Note that only vertices of T_C that start in cliques of height at most $3\Delta N$ are recolored at most $O(\omega^2)$ times to obtain a coloring treated up to C .

Let us now describe the recoloring algorithm and analyze its running time. We root T at an arbitrary node C_r and orient the tree from the root to the leaves. We then do a breadth-first-search starting at C_r and store the height of each node in a table h such that $h[i]$ contains all the nodes of T of height i . Let i_h be the height of T . We apply Lemmas 20 to 23 to every $C \in h[i]$ for i from i_h to 0. Let us show that after step i , the coloring of G is treated up to C for every $C \in h[i]$. It is true for $i = i_h$ since for any $C \in h[i_h]$ the sub-tree T_C of T only contains C . Suppose it is true for some $i > 0$ and let $C \in h[i - 1]$. Let S_1, \dots, S_e be the children of C . For all $j \in 1, \dots, e$, $S_j \in h[i]$ and by assumption the current coloring is treated up to S_j after step i . Thus we can apply Lemmas 20 to 23 to C . After iteration i_h we obtain a coloring of G that is treated up to C_r . Up to adding “artificial” vertices to G , we can assume that C_r is the only clique of T adjacent to a clique path of length $3\Delta N$ (in fact we only need a tuple of $3N$ color vectors) in T and apply Lemmas 20 to 23 until we obtain a coloring such that C_r is canonically colored, and the algorithm terminates. A clique tree of G can be computed in linear time [19], as well as building the table h via a breadth-first-search. Given a clique C , we can access to the cliques of the buffer rooted at T_C in constant time by computing their height and using the table h . Furthermore, a vertex of height i is recolored during the iterations $i + 1, \dots, i + 3\Delta N$ only. As each vertex is recolored at most $O(\omega^2)$ times at each iteration, it follows that the algorithm runs in linear time. Finally, as $N = 3\binom{\omega}{2} + k - \omega + 3$, each vertex is recolored at most $O(\omega^4 \Delta)$ times, and thus the algorithm recolors ϕ to c_0 in at most $O(\omega^4 \Delta \cdot n)$ steps. ◀

The proof of Theorem 2 immediately follows:

Proof of Theorem 2. Let ϕ and ψ be two k -colorings of G with $k \geq d + 4$ and let c_0 be the canonical coloring of G defined in Section 2.1. By Theorem 24, there exists a recoloring sequence from ϕ (resp. ψ) to c_0 of length $O((d + 1)^4 \Delta \cdot n)$. Thus there exists a sequence of length $O_\Delta(n)$ that recolors ϕ to ψ . Furthermore the recoloring sequences from ϕ to c_0 and from ψ to c_0 can be found in linear time by Theorem 24, which concludes the proof. ◀

3.1 Proof of Lemma 20

Recall that in a buffer \mathcal{B} , R_s is the region that sits between the transposition buffer and the color buffer. Before proving Lemma 20, let us first start with some observations.

► **Observation 25.** *Let R be a well-colored region that does not contain color c . If we set $\nu_Q(p) = c$ for every block Q of R , then we obtain a waiting region if R was a waiting region or a color region for the class p , and we obtain a color region if R was a color region for a class $q \neq p$.*

► **Lemma 26 (*)**. *Let (\mathcal{B}, ν) be an almost valid buffer and R_i, R_j be two regions of \mathcal{B} with $1 \leq i \leq j$. Let X be a block in $\{B_i, C_i\}$ and let $Y = B_j$ or $Y = C_N$. Then recoloring (X, \dots, Y, p) with c preserves the continuity property.*

Given a color c , a class p , and a sequence of consecutive blocks (Q_i, \dots, Q_j) , we say that (Q_i, \dots, Q_j, p) is c -free if no vertex of $N[\cup_{t=i}^j Q_t \cap X_p]$ is colored with c . Note that a sequence of (proper) vectorial recolorings of a buffer (\mathcal{B}, ν) that does not recolor A_1 and such that all the classes recolored on C_N are internal to \mathcal{B} yields a (proper) sequence of single vertex recolorings of G by Observation 19. With the definition of clique-tree we can make the following observation:

► **Observation 27.** *Let C be a clique associated with ν_C , S be a child of C , and (\mathcal{B}, ν) be the buffer rooted at S . If $\nu_{C_N}(\ell) \neq \nu_C(\ell)$ for some $\ell \leq \omega$, then the class ℓ is internal to \mathcal{B} .*

Proof. Suppose that the class $\ell \leq \omega$ is not internal to \mathcal{B} . Then there exists a vertex $u \in X_\ell \cap R_N$ which has a neighbor v that does not belong to $R_{N-1} \cup R_N$. Thus u and v must be contained in C or in a clique C' that is an ancestor of C . In the latter case, Property (ii) of clique tree ensures that u must also be contained in C . Then, by definition of ν_C , it must be that $\nu_C(\ell) = \nu_{C_n}(\ell)$. ◀

We also need the following technical lemma:

► **Lemma 28.** *Let (\mathcal{B}, ν) be an almost valid buffer. Let $s < i < N$, c be a color and p be an internal class. If one of the following holds:*

1. R_i is a waiting region, c is a non-canonical color that does not appear in R_s, \dots, R_N and the class p is not involved in a color region, or
2. R_i is a color region for the class p . Moreover c is non-canonical and does not appear in R_s, \dots, R_N , or
3. R_i is a color region for the class p where c is the canonical color that disappears.

Then changing the color of (B_i, \dots, C_N, p) by c also gives an almost valid buffer and a proper coloring of G .

We can now give the flavour of Lemma 20. For a complete proof, the reader is referred to the full version of the article.

Proof of Lemma 20. Let C be the clique associated with vector ν_C and S be a child of C . Let (\mathcal{B}, ν) be the valid buffer rooted at S . Assume that $D_{\mathcal{B}}(\nu_c, \nu) > 0$. Then there exists $p \leq \omega$ such that $\nu_{C_N}(p) \neq \nu_C(p) := c$ and by Observation 27 the class p is internal to \mathcal{B} .

The following sequences of recolorings only recolor blocks of R_s, \dots, R_N and all the recolorings fit in the framework of Lemma 26. Thus Properties 1, 2 and 3 of almost valid buffers are always satisfied. We then only have to check Properties 4 and 5' to conclude the proof. The proof is then based on a case distinction. Let us give a couple of simple cases to give an idea of the general proof:

Case 1. No class is colored with c on R_s, \dots, R_N in ν .

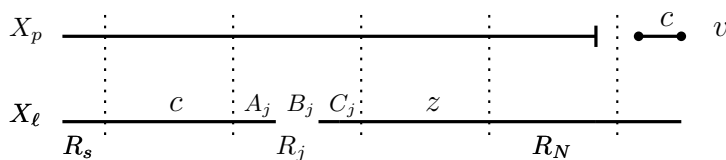
Then c is not canonical since ν_{A_s} is a permutation of the canonical colors by Observation 13. Suppose first that there does not exist a color region for the class p in ν . Since c does not appear in the color buffer, Observation 16 ensures that there exists a waiting region R_i with $s < i < N$. Then by Lemma 28.1, we can recolor (B_i, \dots, C_N, p) with c and obtain an almost valid buffer. Suppose otherwise that there exists a color region R_j for class p in ν . By Lemma 28.2, we can recolor (B_j, \dots, C_N, p) with c and obtain an almost valid buffer. In both cases, the border error decreases.

Case 2. A class $\ell \neq p$ is colored with c , c is canonical, and c disappears in the color region R_j for class ℓ where the non-canonical color z appears (see Figure 4 for an illustration). Let $z' \neq z$ be a non-canonical color and let $c_1 = \nu_{A_s}(p)$. We apply the following recolorings:

- (1) Recolor (B_s, \dots, A_j, ℓ) with z ,
- (2) Recolor (B_s, p) with z' ,
- (3) Recolor (C_s, \dots, C_N, p) with c ,
- (4) Recolor (C_s, \dots, A_j, ℓ) with c_1 .

Recoloring 1 is proper since Observation 15 ensures that the only class colored with z in R_s, \dots, R_N is the class ℓ on (B_j, \dots, C_N) . By the separation property, (B_s, \dots, A_j, ℓ) is z -free in ν . Recoloring 2 is proper as the only non-canonical color in R_s after recoloring 1 is $z \neq z'$. Recoloring 3 is proper as the class p is internal and thus after recoloring 1, (C_s, \dots, C_N, p) is c -free by the separation property. Finally, recoloring 4 is proper as after recolorings 2 and 3, (C_s, \dots, A_j, ℓ) is c_1 -free by the separation property.

Let us show that the resulting coloring defines an almost valid buffer. First note that regions R_{j+1}, \dots, R_N are only modified by recoloring 3 and Observation 25 ensures they remain either waiting or color regions. In particular R_N remains a waiting region. Note that after recolorings 3 and 4, ν' on regions R_{s+1}, \dots, R_{j-1} is obtained from ν by swapping coordinates p and ℓ (on these regions). Thus the nature of these regions is maintained by Observation 11. Since R_j was a color region for the class ℓ and colors c, z in ν , B_j, C_j are not modified and since $\nu'_{A_j}(\ell) = c_1$, R_j is a color region for class ℓ and colors c_1, z in ν' . So the regions R_{s+1}, \dots, R_N remain either waiting or color regions. Furthermore no new color region is created and colors c_1, z are involved in exactly one color region thus Property 4 is satisfied. Finally, R_s is indeed a transposition region in ν' since ν is a valid buffer and z and z' are non-canonical colors, thus Property 5' holds. Furthermore, $\nu'_{C_N}(p) = c$ thus the border error has decreased. ◀



■ **Figure 4** The initial coloring ν for case 2 in the proof of Lemma 20. The rows represent the classes. A blank indicates a color region for the class. The vertical segment at the end of the buffer indicates that the class is internal. The dotted vertical lines separate the different regions.

4 Proof of Lemma 21

The proof distinguishes two cases:

Case 1. there is a region of the transposition buffer of \mathcal{B} that is a waiting region.

The core of the proof consists in iteratively applying the following lemma:

► **Lemma 29 (*)**. *Let (\mathcal{B}, ν) be an almost valid buffer and R_i, R_{i+1} be two consecutive regions with $1 < i < s$ such that R_i is a waiting region and R_{i+1} is a transposition region. Then there exists a recoloring sequence of $R_i \cup R_{i+1}$ such that, in the resulting coloring ν' , R_i is a transposition region, R_{i+1} is a waiting region, and (\mathcal{B}, ν') is almost valid. Moreover only coordinates of $R_i \cup R_{i+1}$ are recolored at most twice.*

Case 2. All the regions of the transposition buffer of \mathcal{B} are transposition regions.

As there are $3\binom{\omega}{2}$ regions in the transposition buffer and only $\binom{\omega}{2}$ distinct transpositions of $\llbracket 1, \omega \rrbracket$, there must exist two distinct regions R_i and R_j with $1 < i < j < s$ for which the same pair of colors is permuted (note that the colors might be associated to different classes in R_i and R_j but it does not matter). The proof consists in applying the following lemma and then applying Case 1.

► **Lemma 30 (*)**. *Let (\mathcal{B}, ν) be an almost valid buffer and c_1, c_2 be two canonical colors. If there exist two transposition regions R_i and R_j where colors c_1 and c_2 are transposed, then there exists a sequence of recolorings of $\cup_{t=i}^j R_t$ such that each coordinate is recolored at most twice, R_i and R_j are waiting regions in the resulting coloring ν' , and (\mathcal{B}, ν') is almost valid.*

References

- 1 Hans-Joachim Böckenhauer, Elisabet Burjons, Martin Raszyk, and Peter Rossmanith. Re-optimization of Parameterized Problems. *arXiv e-prints*, 2018. [arXiv:1809.10578](#).
- 2 M. Bonamy, M. Johnson, I. Lignos, V. Patel, and D. Paulusma. Reconfiguration graphs for vertex colourings of chordal and chordal bipartite graphs. *Journal of Combinatorial Optimization*, pages 1–12, 2012. doi:10.1007/s10878-012-9490-y.
- 3 Marthe Bonamy and Nicolas Bousquet. Token Sliding on Chordal Graphs. In *Graph-Theoretic Concepts in Computer Science - 43rd International Workshop, WG 2017*, pages 127–139, 2017.
- 4 Marthe Bonamy and Nicolas Bousquet. Recoloring graphs via tree decompositions. *Eur. J. Comb.*, 69:200–213, 2018.
- 5 P. Bonsma and L. Cereceda. Finding Paths Between Graph Colourings: PSPACE-Completeness and Superpolynomial Distances. In *MFCS*, volume 4708 of *Lecture Notes in Computer Science*, pages 738–749, 2007.
- 6 Prosenjit Bose, Anna Lubiw, Vinayak Pathak, and Sander Verdonschot. Flipping edge-labelled triangulations. *Comput. Geom.*, 68:309–326, 2018.
- 7 Nicolas Bousquet, Tatsuhiko Hatanaka, Takehiro Ito, and Moritz Mühlenthaler. Shortest Reconfiguration of Matchings. *CoRR*, abs/1812.05419, 2018. [arXiv:1812.05419](#).
- 8 Nicolas Bousquet and Marc Heinrich. A polynomial version of Cereceda’s conjecture. *arXiv e-prints*, 2019. [arXiv:1903.05619](#).
- 9 Nicolas Bousquet and Arnaud Mary. Reconfiguration of Graphs with Connectivity Constraints. In *Approximation and Online Algorithms - 16th International Workshop, WAOA 2018*, pages 295–309, 2018. doi:10.1007/978-3-030-04693-4_18.
- 10 Nicolas Bousquet and Guillem Perarnau. Fast recoloring of sparse graphs. *Eur. J. Comb.*, 52:1–11, 2016.
- 11 L. Cereceda. *Mixing Graph Colourings*. PhD thesis, London School of Economics and Political Science, 2007.

- 12 L. Cereceda, J. van den Heuvel, and M. Johnson. Mixing 3-colourings in bipartite graphs. *Eur. J. Comb.*, 30(7):1593–1606, 2009. doi:10.1016/j.ejc.2009.03.011.
- 13 L. Cereceda, J. van den Heuvel, and M. Johnson. Finding paths between 3-colorings. *Journal of Graph Theory*, 67(1):69–82, 2011. doi:10.1002/jgt.20514.
- 14 Sitan Chen, Michelle Delcourt, Ankur Moitra, Guillem Perarnau, and Luke Postle. Improved Bounds for Randomly Sampling Colorings via Linear Programming. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 2216–2234, 2019. doi:10.1137/1.9781611975482.134.
- 15 R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Heidelberg, third edition, 2005.
- 16 M. Dyer, A. D. Flaxman, A. M Frieze, and E. Vigoda. Randomly coloring sparse random graphs with fewer colors than the maximum degree. *Random Structures & Algorithms*, 29(4):450–465, 2006.
- 17 Carl Feghali. Paths between colourings of graphs with bounded tree-width. *Information Processing Letters*, 144, December 2018. doi:10.1016/j.ipl.2018.12.006.
- 18 Carl Feghali, Matthew Johnson, and Daniël Paulusma. A Reconfigurations Analogue of Brooks’ Theorem and Its Consequences. *Journal of Graph Theory*, 83(4):340–358, 2016.
- 19 Philippe Galinier, Michel Habib, and Christophe Paul. Chordal graphs and their clique graphs. In Manfred Nagl, editor, *Graph-Theoretic Concepts in Computer Science*, pages 358–371, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.
- 20 Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto. Reconfiguration of Maximum-Weight b-Matchings in a Graph. In *Computing and Combinatorics – 23rd International Conference, COCOON 2017, Hong Kong, China, August 3-5, 2017, Proceedings*, pages 287–296, 2017.
- 21 Daniel Lokshantov and Amer E. Mouawad. The complexity of independent set reconfiguration on bipartite graphs. In *Proceedings of the Symposium on Discrete Algorithms, SODA 2018*, pages 185–195, 2018.
- 22 Bojan Mohar and Jesús Salas. On the non-ergodicity of the Swendsen–Wang–Kotecký algorithm on the Kagomé lattice. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(05):P05016, 2010.
- 23 N. Nishimura. Introduction to reconfiguration. *preprint*, 2017.
- 24 A. Suzuki, A. Mouawad, and N. Nishimura. Reconfiguration of Dominating Sets. *CoRR*, 1401.5714, 2014. arXiv:1401.5714.
- 25 J. van den Heuvel. *The Complexity of change*, page 409. Part of London Mathematical Society Lecture Note Series. Cambridge University Press, S. R. Blackburn, S. Gerke, and M. Wildon edition, 2013.

Patching Colors with Tensors

Cornelius Brand

Saarland University (MMCI), Saarland Informatics Campus, Germany
cbrand@mmci.uni-saarland.de

Abstract

We describe a generic way of exponentially speeding up algorithms which rely on Color-Coding by using the recently introduced technique of Extensor-Coding (Brand, Dell and Husfeldt, STOC 2018). To demonstrate the usefulness of this “patching” of Color-Coding algorithms, we apply it *ad hoc* to the exponential-space algorithms given in Gutin et al. (Journal Comp. Sys. Sci. 2018) and obtain the fastest known deterministic algorithms for, among others, the k -internal out-branching and k -internal spanning tree problems. To realize these technical advances, we make qualitative progress in a special case of the detection of multilinear monomials in multivariate polynomials: We give the first deterministic fixed-parameter tractable algorithm for the k -multilinear detection problem on a class of arithmetic circuits that may involve cancellations, as long as the computed polynomial is promised to satisfy a certain natural condition.

Furthermore, we explore the limitations of using this very approach to speed up algorithms by determining exactly the dimension of a crucial subalgebra of extensors that arises naturally in the instantiation of the technique: It is equal to F_{2k+1} , the k th odd term in the Fibonacci sequence. To determine this dimension, we use tools from the theory of Gröbner bases, and the studied algebraic object may be of independent interest.

We note that the asymptotic bound of $F_{2k+1} \approx \phi^{2k} = O(2.619^k)$ curiously coincides with the running time bound on one of the fastest algorithms for the k -path problem based on representative sets due to Fomin et al. (JACM 2016). Here, ϕ is the golden ratio.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Graph algorithms analysis; Computing methodologies → Algebraic algorithms

Keywords and phrases Color-Coding, Extensor-Coding, internal out-branching, colorful problems, algebraic algorithms, multilinear detection, deterministic algorithms, exterior algebra

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.25

Related Version Omitted proofs and details will appear in the author’s thesis [8].

Acknowledgements I want to thank Sarah Berger, Markus Bläser, Radu Curticapean, Holger Dell, Thore Husfeldt, Christian Ikenmeyer, Fahad Panolan, Petteri Kaski, Mikko Koivisto and Meirav Zehavi for helpful discussions and valuable suggestions. I also thank the anonymous referees for their proposed improvements of the manuscript.

1 Introduction

The research in parameterized algorithms has brought forth a vast toolbox of techniques to tackle an ever-growing list of hard computational problems, and provided solutions for those problems in a multitude of shapes and flavors. Recently, Brand, Dell and Husfeldt [9] added another item to the box: A technique called *Extensor-Coding*, based on the properties of the so-called *exterior algebra*, a fundamental object in multilinear algebra.

Extensor-Coding allows us to understand a variety of key approaches in the design of parameterized algorithms in a common language, including the celebrated and by now classic *Color-Coding*-approach of Alon, Yuster and Zwick [1], the *algebraic fingerprints* due to Koutis [24] and Koutis and Williams [25], the representative families from Fomin et al. [19, 18], and the labeled walks of Björklund et al. [5].



© Cornelius Brand;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 25; pp. 25:1–25:16



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

While this yields unification on a conceptual level, the main technical advance made in [9] was with respect to the approximate counting of solutions in the longest path (or k -path) problem. Despite the fact that the technique suggests a straightforward algebraic and *deterministic* algorithm for the k -path problem, its running time was only very slightly (infinitesimally, in fact) below the older bound achieved by Chen et al. [10], and thus at significant distance from Zehavi’s current record bound [36]. In this paper, we will identify several problems where Extensor-Coding *does* improve over the state-of-the-art. To this end, we start from a novel and natural variant of the ubiquitous multilinear detection problem on polynomials computed by arithmetic circuits that is more general than the very well-studied case of cancellation-free circuits. This leads us to a rather generic approach of patching, if you will, algorithms involving Color-Coding, by replacing the Colors used in the algorithm by Extensors, i.e., elements of the exterior algebra.¹ This then yields an exponential speed-up. While this improvement is certainly incremental in nature, the novelty of the approach laid out in this work is of a more conceptual nature. This is discussed further below.

On the other hand, we demonstrate an algebraic barrier, limiting the potential progress that can be made by applying Extensor-Coding in a more or less straightforward manner – just as we do it in this paper for the considered algorithmic problems. This barrier is based on the dimension of a certain subalgebra of the exterior algebra, which presents a lower bound for the cost of computation in the algebra and hence for the entire algorithm. It is noteworthy that the resulting bound, which is exactly the $(2k + 1)$ th Fibonacci number on an underlying vector space of dimension k , asymptotically coincides with the square of the golden ratio, which also appears in the approach of Fomin et al. to the k -path problem, which is based on the very different, purely combinatorial method of representative families [19]. One might argue that this connection is not entirely surprising, since the approaches can be used to solve the same problem and the representative families in fact have their origin in exterior algebra. This, however, is an argument that only bears on a superficial level: It is by no means clear how the very algebraic object studied in this article relates in any way to the use of the exterior algebra in the representative families-approach, and indeed, one can consider uncovering such a precise connection as an intriguing avenue for further research. Therefore, one may regard studying this algebraic object as a first step to tighten the ties between combinatorics and algebra in the study of rather general parameterized algorithms, and not just for the k -path problem.

1.1 Contributions

1.1.1 New and Improved Algorithms

The algorithmic problems we study are the following: The k -*internal out-branching* (k -*IOB*) problem asks for the existence of an out-branching (also called a directed spanning tree or arborescence, among others) with at least k internal, i.e., non-leaf, nodes. The k -*internal spanning* (k -*IST*) tree problem is formulated analogously for undirected graphs. A subset of edges in an edge-colored graph is k -*colorful* if it contains edges of at least k distinct colors, and the definitions of the problems k -*colorful perfect matching* and k -*colorful out-branching* are self-evident.

Our algorithmic advances in these problems are borne by progress in another domain: The k -*multilinear detection* (k -*MLD*) problem has as an input a multivariate polynomial represented through an arithmetic circuit, and asks whether the polynomial contains a

¹ Extensors are often also referred to as antisymmetric tensors, hence the title.

monomial of degree k in which no variable appears with degree more than one. We focus our attention on the restriction of the problem to those instances where a multilinear monomial may only appear with positive coefficient. Since there is no known way to efficiently test this property, this turns the problem into a promise problem. The promise, however, is typically satisfied in combinatorial applications, where the studied polynomials are usually *multivariate generating functions* of the sought combinatorial objects. The decisive subtlety here is that, while the *polynomial* that is computed by the input circuit may not have negative coefficients in its multilinear part, the circuit *itself* may very well contain negative constants and make use of cancellations, and, in particular, it need *not* be monotone. We will make heavy use of this property in the above application problems, and the above point is decisive here for the following reason: These problems can be expressed using determinantal generating functions, and by a theorem of Jerrum and Snir, determinants do *not* have monotone circuits of subexponential size, such that cancellations are actually crucial for their efficient computation. To the best of our knowledge, this is the first fixed-parameter tractable algorithm for the problem.

We prove the following deterministic, exponential-space² record time bounds, and defer the reader to Sect. 4 for a formal statement of the theorem.

► **Theorem 1 (Informal).** *There are deterministic algorithms to solve*

1. *the k -internal out-branching problem and the k -internal spanning tree problem in time $3.21^k \cdot \text{poly}(n)$, and*
2. *the k -colorful perfect matching problem on planar graphs and the k -colorful out-branching problem in time $4^k \cdot \text{poly}(n)$.*

Furthermore, there is a deterministic algorithm that solves the restriction of the multilinear detection problem to circuits computing polynomials with positive coefficients in their multilinear part (as laid out above) in time $4^k \cdot \text{poly}(n)$ on skew arithmetic circuits, and in time $2^{\omega k} \cdot \text{poly}(n) < 5.19^k \cdot \text{poly}(n)$ on general circuits, where ω is the exponent of matrix multiplication.

► **Remark 2.** The bound of 5.19 is not competitive; indeed it is easy to prove that an exponential basis of 4.312 can be obtained using a derandomization of Color-Coding, and Pratt [28] gives a randomized algorithm achieving 4.075. Note, however, that our bound depends on ω , and one can make the point (albeit moot in the foreseeable future) of this dependency making our bound potentially competitive.

► **Remark 3.** Skewness, i.e., the syntactic restriction of each multiplication gate having an input as an operand seems rather strong at a first glance. At a second glance, this impression does not hold up: Without concerning ourselves with the technicalities of algebraic complexity theory, suffice it to say that the polynomials that can be computed by efficient skew circuits are precisely those that are efficient projections of determinants, and determinantal generating functions are known for a variety of combinatorial objects. Equivalently, they are those polynomials that are computed by efficient algebraic branching programs (which is a widely studied and very natural computational model).

The algorithms for the problems of detecting a k -internal as well as k -colorful out-branching (and spanning tree) are established, as demonstrated by Björklund et al. , via the Directed Matrix-Tree Theorem [7]. We reuse the meticulous and very careful analysis of the

² We are aware of the doubtful practical usefulness of exponential-space algorithms in some settings and ask the skeptical reader to think of our results as being motivated theoretically.

parallel monomial sieving technique by Gutin et al. [22]. We can relatively easily replace the employed pseudorandom objects by extensors. This suggests a rather generic way in which to speed up algorithms based on Color-Coding by instead using Extensor-Coding.

As far as the k -colorful planar matching problem is concerned, as in [22], we use a Pfaffian computation. However, we cannot perform the square root extraction that is necessary to make use of the determinantal identities for the Pfaffian, but rely on a result of Flarup et al. [15] for a direct computation of Pfaffians with skew arithmetic circuits.

Let us conclude the discussion of our algorithmic contributions with a comment on their relevance. As far as combinatorial problems are concerned, the obtained technical improvements are incremental. However, focusing on these individual results is, in a way, a red herring: The important point is that we can obtain speed-ups in sophisticated algorithms for well-studied problems in a simple, *ad hoc* manner by replacing colors with extensors. The only requirement here is the, as argued, rather mild one of the combinatorial objects in question being described through a determinantal formula. This is not primarily a technical, but a *conceptual* insight that is meant to guide the design of deterministic exponential-space algorithms in the future, and to encourage revisiting existing bounds using extensors. To demonstrate how to go about this, we do so for k -IOB, a very well-studied problem where the current state-of-the-art was obtained through a sequence of incremental improvements, each more involved than the previous one.

A similar case can be made for the monomial detection algorithms, which are obtained in an admittedly straightforward manner by an application of the Extensor-Coding method, which was certainly available at the time of writing of [9]. The takeaway here, however, is that polynomials with positive coefficients computed by efficient skew circuits are not some arbitrary, but instead very natural class of polynomials, namely those that arise as determinantal generating functions for combinatorial objects, which are of central interest in algebraic combinatorics.

1.1.2 Algebraic Limitations

Broadly speaking, when aiming towards deterministic decision algorithms, Extensor-Coding essentially works by evaluating a multivariate polynomial associated with the input over an exterior algebra. This algebra is of dimension 4^k , where k will typically be the parameter of the input instance (while it formally is half the dimension of the underlying vector space).

In this way, it is similar to a method introduced by Koutis [24]. It differs, however, in the points at which the polynomial associated with the input instance is evaluated. While Koutis, and later, Williams [33], rely on random evaluation points, in [9], certain carefully constructed vectors are used. One can readily observe that “evaluating a polynomial” involves, on an arithmetic level, nothing but multiplications and additions. In particular, if one evaluates a polynomial over any algebraic structure that is closed under multiplication and addition, one will always obtain again an element of this algebraic structure after evaluation. Turning this around, one can always restrict one’s attention to the closure of (i.e., the substructure generated by) the set of evaluation points. In particular, it might be far easier to actually implement the arithmetic operations only over this substructure than over the entire structure.

In this paper, we will examine the subalgebra generated by the aforementioned special evaluation points (i.e., the smallest set closed under addition, scaling with a field constant, and multiplication containing all the evaluation points). Note now that the dimension of an algebra provides a trivial lower bound on the cost of general computation in it, simply because one has to write down its elements’ coordinates at some point. Conversely, the dimension itself can be used to derive a trivial upper bound for the cost of computation as

well, just by a look-up of the structural constants of the algebra, but this bound is generally far from optimal. For example, naive multiplication of degree- d polynomials (which can be modeled as objects of a $O(d)$ -dimensional algebra over some field) takes around $O(d^2)$ field operations, while (over compatible fields) the Fast Fourier Transform method yields the classic bound of $O(d \log d)$. A running time of the latter form, i.e., quasilinear in the dimension of the algebra, is of course the best one can hope for. Note however that, in general, it is a highly nontrivial, and sometimes impossible, task to come up with such fast multiplication algorithms in any kind of algebraic structure.

We prove that, surprisingly (for reasons that will be expanded on later), the dimension of the subalgebra generated by the evaluation points used in Extensor-Coding is of dimension *exponentially smaller* than 4^k , i.e., the dimension of the full algebra. At first, this seems to open up a tantalizing new point of attack on one of the most prominent open problems in the area of parameterized algorithms: To exhibit a *deterministic* algorithm for the k -path problem that matches the running time of $2^k \cdot \text{poly}(n)$ for the best *randomized* algorithms [33, 24]. Now, *a priori*, we could hope for the studied subalgebra to be of dimension 2^k (but not much smaller, by a result of Koutis-Williams [26]). Then, a quasilinear multiplication algorithm would give a bound of $2^k \cdot \text{poly}(k)$ field operations, and (assuming all coefficients stay of moderate size) hence produce a deterministic algorithm solving the problem in time $2^k \cdot \text{poly}(n)$.

Unfortunately, we share the fate of Tantalos:

► **Theorem 4 (Informal).** *The dimension of the subalgebra of the exterior algebra over the complex vector space of dimension $2k$ generated by the evaluation points used in Extensor-Coding is of dimension exactly F_{2k+1} , which is the $(2k + 1)$ th Fibonacci number, and this is asymptotically bounded as $2.618^k \leq F_{2k+1} \leq 2.619^k$.*

Again, it will be stated formally in Sect. 5. Somewhat unluckily, this does also not suffice to improve the state-of-the-art for k -path, even if we had a sufficiently fast multiplication algorithm, since Zehavi's [36] intricate algorithm has an exponential basis of 2.597 in its running time.

This result can be interpreted in two ways: On the one hand, as laid out above, it can be seen as a negative result, ruling out one approach for solving the k -path problem deterministically and fast. On the other hand, while a quasilinear multiplication algorithm for the subalgebra might be a lot to hope for, even an algorithm using, say, $2.619^{\omega k/2} < 3.136^k$ field operations would mean significant improvements for all the problems studied in this paper, including in particular deterministic multilinear detection for a subclass of circuits with cancellations. It is not at all clear how to implement multiplication over this algebra faster than trivially, and we leave this as a challenging open problem.

Let us comment on the bound of $2.619^{k\omega/2}$, which might at first seem arbitrary. This is simply the bound one would obtain from a matrix representation of dimension at most d , i.e., having dimension $\sqrt{2.619^{k/2}} \times \sqrt{2.619^{k/2}}$. For this reason, it appears in the recent breakthrough work of Umans [32] and Ching-Yun Hsu and Umans [12] on generalized Fast Fourier Transforms on arbitrary finite groups. Therefore, our result nurtures the hope for even further, significant improvement for a variety of parameterized problems, conditional on the development of fast multiplication algorithms over this algebraic structure.

Furthermore, we exhibit a whole family of subalgebras that enjoy all the desired properties of the subalgebra that is used in [9], and that obey the same upper bound on their dimension, while the lower bound argument does not easily carry over, and it is an exciting possibility that, somewhere among these subalgebras, there is one that has even lower dimension.

As for our algorithmic contributions, let us comment on the relevance of the algebraic barrier, and in particular, why this highly special algebraic object is of wider interest. Given the apparent connection to the work of Fomin et al. as well as the recent insights of Pratt [28], the study of algebraic approaches may have much closer ties to the classic combinatorial techniques than is currently visible. Establishing the dimension bounds as done here is clearly necessary and important in order to uncover these ties; ties which, eventually, might make the study of parameterized algorithms and their limitations appealing to a more diverse audience even outside of parameterized algorithms and complexity, in a way similar (even if not as ambitious by far) to how algebraic geometers found interest in complexity theory through its geometric reformulation. Furthermore, as stated, exhibiting fast multiplication algorithms for this problem would directly imply massive speed-ups for problems that are amenable to be “patched” using extensors.

1.2 Related work

The k -internal out-branching and spanning tree problems have attracted a significant amount of attention over the last years [17, 22, 6, 7, 27, 38, 37, 16, 13, 11, 21, 14, 29]. The current deterministic record bounds for all the aforementioned graph problems were recently given in Gutin et al. [22], using monomial sieving in combination with Color-Coding and suitable pseudorandom objects. The bounds they obtain (in the exponential-space setting) for the k -internal out-branching and spanning tree problems are $3.41^k \cdot \text{poly}(n)$, and $4.32^k \cdot \text{poly}(n)$ for the k -colorful perfect matching and out-branching problems.

The detection of k -multilinear terms in polynomials computed by arithmetic circuits lies at the heart of the design of the fastest randomized algorithms for a host of parameterized problems, such as the longest path problem on directed graphs, the k -tree problem, the t -dominating set problem and the m -dimensional k -matching problem, with a record bound of $2^k \cdot \text{poly}(n)$ for randomized k -multilinear detection [26, 33]. The crux is that, for the arithmetic circuits in these algorithms, it is required that they be *monotone*, i.e., do not involve any cancellations of terms. On this class of monotone arithmetic circuits, the k -multilinear detection problem can be solved deterministically in time $3.85^k \cdot \text{poly}(n)$, using the combinatorial notion of representative sets [18].

Recently, the first fixed-parameter tractable *randomized* algorithms were developed for the problem on general arithmetic circuits [9], which has sparked further work in the area, announcing a polynomial-space version and $4.08^k \cdot \text{poly}(n)$ as a new record bound on general circuits [3, 2, 28].

Due to the particularity of the algebraic object studied for the dimension barrier, it is highly doubtful whether any work directly related to it has been done – at the least, we are not aware of such work. However, the idea of proving dimension lower bounds in order to limit the use of certain parameterized algorithmic techniques was already present in work by Koutis and Williams [26]. Their work, however, gives a more general lower bound and relies on entirely different technical ideas, making it akin to our work only in a broader conceptual sense.

Omitted proofs and details will appear in [8], which we call the full version.

2 Preliminaries

In what follows, $2 \leq \omega < 2.374$ [30, 34] is the exponent of matrix multiplication.

An *arithmetic circuit* is a directed acyclic graph with a single vertex of out-degree 0, called the *output*, a set of vertices with in-degree 0 which are labeled with variables or complex numbers, called the *inputs* and labels $+$ and \times on all vertices that are not inputs.

A vertex labeled with $+$ or \times is, respectively, called a $+$ -gate or \times -gate. All \times -gates are required to have in-degree at most two. We call an arithmetic circuit *skew* if, for all \times -gates, at least one of the arcs ending at the gate comes from an input. The *polynomial computed by an arithmetic circuit* is defined in the obvious inductive way: It is equal to the label of the inputs, and for a gate that is not an input, it is defined as the result of the operation indicated by the label of the gate, applied to its two inputs.

Given a directed graph $D = (V, A)$, we call a subgraph B of D *out-branching* if B is an oriented tree with exactly one vertex r of in-degree zero, the *root*, and B is spanning. Vertices of out-degree zero are called *leaves*, and vertices that are not leaves are *internal*. For an integer k , an out-branching is *k-internal* if it contains at least k internal vertices.

A set of arcs is called a *matching* if no vertex of V is contained in two arcs, and a matching is *perfect* if every vertex is contained in some arc of the matching.

For an edge-colored graph G , directed or undirected, we call a subgraph S of G *k-colorful*, if the edge set of S contains at least k differently colored edges.

2.1 A Review of Extensor-Coding

Although not a strict prerequisite, some knowledge of the technique and its uses for the longest path problem make the following certainly more digestible. An elementary exposition of the employed algebraic objects can be found in the original work [9]. However, we will demonstrate the technique in the appropriate brevity. This should, in principle, suffice to understand everything that happens here. For a more mathematically concise and thorough treatment of the algebraic background material, we encourage the interested reader to consult any given textbook on algebra, for example the one by Birkhoff and Mac Lane [4].

Let us first review some basic algebraic notions. A complex unital associative *algebra* is a ring A that is simultaneously a complex vector space, such that scalar multiplication is compatible with the multiplication in the ring. That is, for any $\lambda \in \mathbb{C}$ and $a, b \in A$, it holds that $\lambda(ab) = (\lambda a)b = a(\lambda b)$ and there is an element $1 \in A$ with $1a = a = a1$ for all $a \in A$. Since complex unital associative algebras are the only types of algebras we will encounter henceforth, we understand all these properties whenever we speak merely of an algebra. The dimension of an algebra A is the dimension of A as a complex vector space.

2.1.1 The Exterior Algebra

Let V be the complex vector space \mathbb{C}^k , endowed with its canonical basis $\{e_1, \dots, e_k\}$. We can now consider the set of formal, *non-commutative* polynomials $\mathbb{C}\langle e_1, \dots, e_k \rangle$ that can be formed in the “indeterminates” (or *generators*) e_1, \dots, e_k .

Let us define a kind of multiplication on these generators that is denoted as \wedge , called the *wedge product*. The wedge product operation on the generators is defined to satisfy the anti-commutativity relation: $e_i \wedge e_j = -e_j \wedge e_i$ for all $1 \leq i, j \leq k$. For $i = j$, this means that $e_i \wedge e_i = -e_i \wedge e_i$, which over \mathbb{C} implies $e_i \wedge e_i = 0$ for all i .

By repeatedly applying this rule, we can understand the multiplication of more than two generators as follows. Let $1 \leq i_1, \dots, i_t \leq k$ be natural numbers, and consider the wedge product $e_{i_1} \wedge \dots \wedge e_{i_t}$. Exhaustive application of the rules now implies that this product becomes zero if $i_j = i_{j'}$ for any distinct j, j' . Otherwise, it is equal to $\text{sgn}(\sigma)e_{i_{\sigma(1)}} \wedge \dots \wedge e_{i_{\sigma(t)}}$, where σ is the permutation that brings the sequence i_1, \dots, i_t in ascending order.

This multiplication is extended to linear combinations of wedge products of the generators by distributivity and bilinearity. The vector space generated by all such wedge products of generators is called the *exterior algebra* over V and is denoted by $\Lambda(V)$. The vector space $\Lambda(V)$ is turned into an algebra by equipping it with the wedge product as multiplication. We have seen that any reordering of the factors either leaves the original wedge product intact

or leads to a sign change. Hence, the set of ordered wedge products $\{e_{i_1} \wedge \cdots \wedge e_{i_t} \mid 0 \leq t \leq k, i_1 < \cdots < i_t\}$ form a linear basis of $\Lambda(V)$. This shows that $\Lambda(V)$ is of dimension 2^k . For any set $S \subseteq \{1, \dots, k\}$, we write e_S to denote $e_{i_1} \wedge \cdots \wedge e_{i_{|S|}}$, where the elements of S are assumed to be in ascending order, i.e., $i_1 < \cdots < i_{|S|}$. We remark that we can regard V as a linear subspace of $\Lambda(V)$ via a vector's basis representation, i.e., $(x_1, \dots, x_k)^T \in \mathbb{C}^k$ corresponds uniquely to $x_1 e_{\{1\}} + \cdots + x_k e_{\{k\}}$.

Let us quickly comment on the cost of computation in $\Lambda(V)$. It is clear that addition of two elements of $\Lambda(V)$ can be performed using 2^k arithmetic operations, namely by pointwise addition, over \mathbb{C} . Clearly, this can be performed in time polynomial in the length of the basis coefficients of the two summands.

For a restricted but crucial variant of multiplication in $\Lambda(V)$, called *skew multiplication*, we recall the following observation:

► **Proposition 5** ([9], Sect. 2.3). *Given an element $x \in \Lambda(\mathbb{C}^k)$ and $y \in \mathbb{C}^k \subseteq \Lambda(\mathbb{C}^k)$ as a list of basis coefficients, their product $x \wedge y \in \Lambda(\mathbb{C}^k)$ as a list of basis coefficients can be computed using $2^k \cdot \text{poly}(k)$ arithmetic operations over \mathbb{C} .*

Additionally, if the bitlength of coefficients of x and y is bounded in by τ , then their product can be computed in $2^k \cdot \text{poly}(\tau)$ bit operations.

As for general multiplication, let us record what Włodarczyk [35] showed implicitly:

► **Theorem 6** ([35]). *Given two elements $x, y \in \Lambda(\mathbb{C}^k)$ as a list of basis coefficients, their product $x \wedge y \in \Lambda(\mathbb{C}^k)$ as a list of basis coefficients can be computed using $2^{\omega k/2} \cdot \text{poly}(k)$ arithmetic operations over \mathbb{C} .*

Additionally, if the bitlength of coefficients of x and y is bounded by τ , then their product can be computed in $2^{\omega k/2} \cdot \text{poly}(\tau)$ bit operations.

Since the connection to the exterior algebra in [35] is not made explicitly, we give a self-contained (and somewhat simpler) proof of Włodarczyk's result, adapted to our terminology, in the full version.

► **Remark 7.** The best known way to compute a general wedge product is due to insights of Włodarczyk: This goes by reducing the wedge product to the product in a so-called Clifford algebra, which is an algebra that, filtered by degree, gives rise to the exterior algebra as its associated graded algebra. This is surprising from a mathematical perspective: The exterior algebra is an especially degenerate Clifford algebra, and yet the latter is used to compute in the former. Finding other, possibly faster approaches for the computation of wedge products is an independent research direction that we feel worth pursuing in the future, given that the wedge product is the epitomic operation in all of multilinear algebra.

An easy, but fundamental standard observation is that, for any $t \leq k$, the t -factor wedge multiplication map

$$V^t \rightarrow \Lambda(V), \quad x_1, \dots, x_t \mapsto x_1 \wedge \cdots \wedge x_t \in \Lambda(V) \tag{1}$$

can be written down in coordinates as the determinants of the $t \times t$ -minors of the matrix $(x_1 \mid x_2 \mid \dots \mid x_t)$ obtained as the juxtaposition of x_1, \dots, x_t :

$$x_1 \wedge \cdots \wedge x_t = \sum_{S \in \binom{[k]}{t}} \det(x_1 \mid x_2 \mid \dots \mid x_t)_S e_S,$$

where $(x_1 \mid x_2 \mid \dots \mid x_t)_S$ is the $t \times t$ -minor of the matrix $(x_1 \mid x_2 \mid \dots \mid x_t)$ indexed at the rows S . In particular, this entails that the k -fold wedge product is just the determinant map:

$$x_1 \wedge \cdots \wedge x_k = \det(x_1 \mid x_2 \mid \dots \mid x_k) e_{[k]}.$$

2.1.2 Lifts

Consider the direct sum of vector spaces $V \oplus V \cong \mathbb{C}^{2k}$, and let $\iota_1, \iota_2 : V \rightarrow V \oplus V$ be the canonical embeddings of V into $V \oplus V$ as the left and right summand, respectively. In coordinates, this corresponds respectively to pre- or appending k zeros to a vector. Consider the *lift mapping* $\Lambda(V) \rightarrow \Lambda(V \oplus V)$, $x \mapsto \bar{x} := \iota_1(x) \wedge \iota_2(x)$. We now have (see also [9, Sect. 3.4]) the following.

$$\bar{x}_1 \wedge \cdots \wedge \bar{x}_k = \pm \det(x_1 \mid \cdots \mid x_k)^2 e_{[2k]},$$

where the sign only depends on k .

2.1.3 Vandermonde Codings

Determinants vanish on singular matrices. To prevent unwanted vanishing of determinants later on, we will use a set of vectors that are, in a sense, maximally linearly independent: Vandermonde vectors. To this end, consider the mapping

$$\phi : \mathbb{C} \rightarrow V, c \mapsto (1, c, c^2, \dots, c^{k-1})$$

and its lifted variant

$$\bar{\phi} : \mathbb{C} \rightarrow V \oplus V, c \mapsto \overline{\phi(c)}.$$

This has the nice property that, on the image of ϕ (or $\bar{\phi}$ for that matter), the k -fold wedge product map as in Eq. (1) is zero only when two factors are *equal*. In particular, the mapping $\mathbb{C}^k \rightarrow \Lambda(V \oplus V)$, $(c_1, \dots, c_k) \mapsto \bar{\phi}(c_1) \wedge \cdots \wedge \bar{\phi}(c_k)$ is zero exactly on the set of points that have at least two coordinates equal. Indeed, the coordinates of this map witness this in the clearest way possible: It is the well-known Vandermonde-determinant $\prod_{i < j} (c_i - c_j)$. An analogous statement of course holds for $\bar{\phi}$.

3 Monomial Detection Problems

As a kind of warm-up, we will give a rather direct, but very useful first application to a special kind of monomial detection problem. The general problem presents itself as follows: As input, it obtains a multivariate (and now again commutative) polynomial $f \in \mathbb{C}[X_1, \dots, X_n]$ in n indeterminates. Our task is now to decide whether or not f contains a monomial of degree k such that no indeterminate appears twice. A variation of this is to ask whether f contains a monomial such that at least k distinct indeterminates appear in it. The degree of hardness of this problem obviously hinges on the way in which f is represented. When f is given in its sparse representation as a list of monomials and coefficients, then deciding this question is trivial.

This not the case, however, if f is represented by an arithmetic circuit, which is the setting we are interested in. Indeed, we are interested only in a semantically defined subclass of arithmetic circuits: Those that compute a polynomial f such that every multilinear monomial that appears in f does so with positive coefficient. As stated before, it is crucial to note that this does *not* mean a monotonicity restriction for the input circuit, and it may well involve cancellations of terms and negative constants. Let us formally define the set of circuits we are interested in:

► **Definition 8.** *Let C be an arithmetic circuit that computes a polynomial f . We call C combinatorial if f has non-negative coefficients on its multilinear part, and C can be evaluated over \mathbb{Z} at numbers of absolute value at most τ using $\text{poly}(\tau)$ bit operations.*

We remark that the last condition of this definition is a barely concealed crutch to avoid having to think about subtleties regarding a possible doubly exponential blowup of inputs in general arithmetic circuits, which are irrelevant in our applications. For a very similar reason, we only speak about evaluation over \mathbb{Z} ; namely, in order to ignore potential issues with representations of complex numbers.

3.1 Multilinear Detection

As promised, we will now start out with an easy application of Extensor-Coding. Speaking of promises, it is again in order to remark that the problems discussed henceforth are promise problems, in the sense that there is no known efficient method of checking whether an input circuit satisfies the condition of being combinatorial.

► **Theorem 9.** *There is a deterministic algorithm that, given a combinatorial arithmetic circuit C of size s and an integer k , decides whether or not the polynomial computed by C contains a multilinear monomial of degree k in time $2^{\omega k} \cdot \text{poly}(s) < 5.19^k \cdot \text{poly}(s)$.*

Proof. Consider the lifted Vandermonde coding $\bar{\phi}$, and assume that the polynomial computed by C is n -variate.

The algorithm then simply evaluates C at $(\bar{\phi}(1), \dots, \bar{\phi}(n))$, and outputs “yes” if and only if the coefficient of $e_{[2k]}$ in the resulting element of $\Lambda(V \oplus V)$ is non-zero.

The running time is immediate from the definition and Theorem 6, and correctness can be seen as follows. We first have to take care of the fact that our algebra is not commutative, strictly speaking, but nevertheless we evaluate a commutative polynomial over it. This is remedied either by a standard degree- k homogenization argument on C , and, depending on k , a subsequent single sign correction of the result. Alternatively, one can observe that the signs are consistent across each degree individually, and monomials of different degrees are linearly independent, so that the different signature arising when evaluation over the image of $\bar{\phi}$ will not make a difference. From the fact mentioned in discussion about Vandermonde codings, every monomial containing an indeterminate twice will vanish. The parts of degree less than k do not enter into the coefficient of $e_{[2k]}$, and parts of degree more than k will go to zero anyways. Now, let L be the set of multilinear monomials in the polynomial computed by f . Each such monomial m identifies a subset of $\{1, \dots, n\}$, and by abuse of notation, we will not distinguish between a monomial and a subset. Furthermore, we denote with c_m the coefficient of m , which is non-negative by the assumption on C , and let V_m be the $2k \times 2k$ matrix $(\bar{\phi}(i))_{i \in m}$. Then the coefficient of $e_{[2k]}$ can be seen to be equal to $\sum_{m \in L} c_m \cdot \det(V_m)^2$, which is non-zero if and only if one of the determinants is non-zero. This in turn happens if and only if there is a multilinear monomial of degree k in the polynomial computed by C . ◀

We also obtain the more useful skew variant:

► **Theorem 10.** *There is a deterministic algorithm that, given a skew combinatorial arithmetic circuit C of size s and an integer k , decides whether or not the polynomial computed by C contains a multilinear monomial of degree k in time $4^k \cdot \text{poly}(s)$.*

Proof. Follows verbatim like Theorem 9 after replacing Theorem 6 by Proposition 5. ◀

3.2 k -Distinct Detection

We will now turn to the monomial detection problem that will later on be used in applications, namely the k -distinct detection problem. Again, the input here is an arithmetic circuit, but this time, the task is to decide whether there exists a monomial containing at least k distinct

indeterminates. Using the folklore trick of replacing every variable x_i by $1 + t \cdot x_i$ with a formal indeterminate t , turning a nilpotent variable x_i into an (almost) idempotent one, and then extracting the coefficient of t^k , we obtain:³

► **Theorem 11.** *There is a deterministic algorithm that, given a skew combinatorial arithmetic circuit C of size s that computes a polynomial that only has non-negative coefficients, and an integer k , decides whether or not the polynomial computed by C contains a monomial with at least k different variables in time $4^k \cdot \text{poly}(s)$.*

A proof sketch is given in the full version. Equipped with these observations, we may now turn to our application problems.

4 Graph Problems

We will make use of the classic Directed Matrix-Tree Theorem, following the presentation in [7]. Let us first define the *Laplacian* of a directed graph $G = (D, A)$. To this end, let $X = \{x_a \mid a \in A\}$ be a set of formal indeterminates labeled with the arcs of a graph, and define the matrix $L = (\ell_{uv})_{u,v \in V}$ through

$$\ell_{uv} = \begin{cases} \sum_{w \in V: wu \in A} x_{wu} & \text{if } u = v \\ -x_{uv} & \text{if } uv \in A \\ 0 & \text{if } uv \notin A \end{cases}.$$

After fixing a root $r \in V$, we will consider L_r , the *Laplacian punctured at r* , which is defined as the matrix obtained from L by striking row r and column r . With these definitions in place, we have the following well-known theorem, and just as [7], we refer to the corresponding chapter of Gessel and Stanley in the Handbook of Combinatorics [20] for a proof.

► **Theorem 12 (Directed Matrix-Tree Theorem).** *Let $G = (D, A)$ be a directed graph. For all $r \in V$, the following holds.*

$$\det L_r = \sum_{\substack{T = (V, B) \text{ is an} \\ \text{out-branching of } G \\ \text{rooted at } r}} \prod_{b \in B} x_b.$$

In other words, the determinant of L_r is the multivariate generating function of the set of out-branchings rooted at r . The important insight is now the following: All known (randomized) efficient algorithms for detecting k -multilinear terms – and, by extension, k -distinct terms – in the polynomial computed by an arithmetic circuit rely on this circuit not involving cancellations in their computation, i.e., they need to be *monotone*.

However, by a theorem of Jerrum and Snir [23], computing $\det L_r$ using such a monotone circuit requires circuits of *exponential size* in n .

On the other hand, there are efficient skew arithmetic circuits (this time with cancellations) for computing the $n \times n$ determinant polynomial:

► **Theorem 13 ([31]).** *There is a family of skew arithmetic circuits $(C_n)_{n \in \mathbb{N}}$ such that C_n computes the $n \times n$ determinant polynomial, and the size $s(n)$ of C_n satisfies $s(n) \leq \text{poly}(n)$. Furthermore, there is an algorithm that, upon input 1^n , outputs a description of C_n in time $\text{poly}(n)$, and every circuit can be evaluated over \mathbb{Z} in polynomial time in the length of the input representation.*

³ We might just replace the indeterminate t by 1, and extract the degree- k term of the result in the exterior algebra. Using an extra indeterminate t allows us to avoid explaining how *degree* is a well-defined concept even over exterior algebras.

In fact, the $n \times n$ determinant polynomial is complete – for a suitable notion of reduction – for the adequately named class VDET of polynomial families computable by $\text{poly}(n)$ -sized skew arithmetic circuits, defined *ibid.* Importantly, this together with the Matrix-Tree Theorem 12 also shows that $\det L_r$ is a combinatorial polynomial.

4.1 Out-Branchings

Let us now proceed gently with a first application of what we have gathered so far.

► **Theorem 14.** *There is a deterministic algorithm that, given a directed edge-colored graph D on n vertices and an integer k , decides whether D has a k -colorful out-branching in time $4^k \cdot \text{poly}(n)$.*

Proof. First, replace every variable x_a by a fresh variable corresponding to its color $c(a)$, say $y_{c(a)}$, and denote the corresponding symbolic matrix with $L_r(y)$. Since $\det L_r$ is combinatorial and skew, so is $\det L_r(y)$, and we can perform the k -distinct detection from Theorem 11 in the claimed running time. The existence of a monomial with k distinct variables in $\det L_r(y)$ is now clearly equivalent to the existence of a k -colorful out-branching in D . ◀

Theorems 12, 13 and Theorem 11 immediately yield a deterministic algorithm for the problem, running in time $4^k \cdot \text{poly}(n)$. We note that this is already a significant improvement over the time bound of $5 \cdot 14^k \cdot \text{poly}(n)$ on the runner-up algorithm of [37].

► **Proposition 15** (Superseded by [22]). *There is a deterministic algorithm that, given a directed graph D on n vertices and an integer k , decides whether D has a k -internal out-branching in time $4^k \cdot \text{poly}(n)$.*

Proof. First, replace every variable x_{uv} by a fresh variable corresponding to its tail, say y_u , and denote the corresponding symbolic matrix with $L_r(y)$. Since $\det L_r$ is combinatorial and skew, so is the n -variate polynomial $\det L_r(y)$, and we can perform the k -distinct detection from Theorem 11 in the claimed running time. It has been observed by Björklund et al. [7] that this is neatly equivalent to the input instance containing a k -internal out-branching. ◀

To speed this up, we can more or less just plug in an Extensor-Coding into the analysis from [22] and set a new record bound for the k -internal out-branching problem. We defer the proof to the full version.

► **Theorem 16.** *There is a deterministic algorithm that, given a directed graph D on n vertices and an integer k , decides whether D has a k -internal out-branching in time $3.21^k \cdot \text{poly}(n)$.*

The corresponding results for k -internal spanning trees of undirected graphs follow immediately by standard reductions to the directed case (see [22]).

4.2 Colorful Planar Perfect Matchings

Just like out-branchings, perfect matchings in planar graphs have an efficiently computable multivariate generating function, namely the *Pfaffian* $\text{Pf } A$ of a suitable skew-symmetric matrix A . Gutin et al. [22] employ a determinantal identity for the Pfaffian, and do so by evaluating the determinant polynomial in question over the integers in a black-box fashion. Namely, they exploit that $\text{Pf}(A)^2 = \det(A)$. In the very last step, this requires a square root extraction, which is a perfectly viable path over the integers, but not over more general algebras. Therefore, instead of construing $\text{Pf}(A)$ as $\sqrt{\det(A)}$, we rely on an observation of Flarup et al. [15], who show that the Pfaffian has efficient skew circuits. Putting this together, we immediately obtain:

► **Theorem 17.** *There is a deterministic algorithm that, given an undirected edge-colored planar graph G on n vertices and an integer k , decides whether G has a k -colorful perfect matching in time $4^k \cdot \text{poly}(n)$.*

5 The Dimension Barrier

In this section, we will explain Theorem 4. The entire proof is deferred to the full version.

5.1 The Subalgebra Generated by Evaluation Points

Let us first elaborate on the fundamental insight that motivates this result and that was alluded to already in the introduction: Let f be a complex polynomial, such as the polynomial computed by a circuit in, say, Theorem 11. During the evaluation of this polynomial f (or rather the circuit computing it) at points from the image of $\bar{\phi}$, only sums and wedge products of elements of the image of $\bar{\phi}$ are ever formed. Note that this is what we do in all our applications: We only plug in points of the form $\bar{\phi}(c)$ for some $c \in \mathbb{C}$. We can say this rigorously and concisely by stating that during such an evaluation, all computation may only occur in the *subalgebra of $\Lambda(V \oplus V)$ generated by the image of $\bar{\phi}$* , which is – by definition – the set of all sum-wedge product combinations of the generating set. It is precisely the subalgebra generated by these elements $\bar{\phi}(c)$ that we will study. The fundamental quantity associated with this subalgebra is its dimension, that is, the dimension of the subalgebra as a complex vector space. Let us stress again the argument from the introduction that any potential progress on the technique in its present form hinges on the dimension of this subalgebra: It provides both strict lower bounds and a good guide towards the upper bounds one can hope for when solving problems using Extensor-Coding.

5.2 A Family of Related Subalgebras

The decisive property that makes Vandermonde codings so useful is that any distinct k of them are linearly independent, which is commonly referred to the set of Vandermonde vectors being in general position. This, however, is not only a property enjoyed by Vandermonde vectors. First of all, any finite random set of vectors is in general position almost certainly.

This suggests that instead of considering the subalgebra generated by Vandermonde codings, one might as well just take any n random vectors and proceed with them. Extensive computational experiments have shown, however, that the expected dimension of this subalgebra is almost as high as the dimension of the full algebra. Curiously, it seems to be equal (and in fact this was the only case that ever occurred during all our experiments) to the k -th Catalan number, which grows asymptotically faster than $4^{(1-\varepsilon)k}$ for all $\varepsilon > 0$.

It is worth pondering about this phenomenon for a little while. From a matroid perspective, the Vandermonde codings just correspond to a representation of the k -uniform matroid over an n -element universe. One might now hope that the dimension of the subalgebra generated by the lifts of the columns of this representation matrix is a matroid invariant; however, this is *not* the case. Even more, most representations are just as bad as the worst case.

Remarkably, the extraordinarily well-behaved case of the Vandermonde representation transfers quite directly to a related family of subalgebras. Consider any set $P = \{p_1, \dots, p_k\}$ of formal univariate polynomials that are linearly independent and of degree less than k . In other words, a basis of the set of polynomials of degree less than k . We can form, for any n , a $k \times n$ evaluation matrix of this set P , where the i -th row is given as $(p_i(1), p_i(2), \dots, p_i(n))$. If we pick as P the standard monomial basis, this is just the Vandermonde representation.

However, we may as well pick any other basis, and the resulting matrix will again have the property that any subset of k columns is linearly independent. This works, provided the p_i do not have a common root at the evaluation point. We can take care of this easily by just picking a different, appropriate set of evaluation points. Note that this set exists (and in fact, almost certainly, any random set will do) by the fact that the p_i are distinct polynomials. Now, we can again study the subalgebra of $\Lambda(V \oplus V)$ generated by the lifts of the column vectors of this evaluation matrix. Surprisingly, the argument for the upper dimension bound carries over immediately, but we could not find a corresponding proof for the lower bound. This opens up the exciting possibility of finding lower-dimensional algebras in this family, that could lead to even faster algorithms. Note that despite the main motivation for studying this algebra may stem from the flagship k -path problem, due to the connection to multilinear detection, finding these algorithms has impacts for all the problems that reduce to this special case of multilinear detection, including those studied in this paper.

5.3 The Barrier

As announced, we will merely state the relevant theorem. A full, detailed proof can be found in the full version.

► **Theorem 18.** *Let $V = \mathbb{C}^k$ and let F_{2k+1} be the $(2k+1)$ th Fibonacci number. The subalgebra of $\Lambda(V \oplus V)$ generated by the image $\{\overline{\phi}(c) \mid c \in \mathbb{C}\}$ of $\overline{\phi}$ is of dimension exactly equal to F_{2k+1} .*

Furthermore, for any linear basis P of the univariate polynomials of degree at most k , denote for $c \in \mathbb{C}$ with $P(c)$ the column vector obtained by evaluating all polynomials in P at c . Then, the subalgebra generated by $\{\overline{P}(c) \mid c \in \mathbb{C}\}$ is of dimension at most F_{2k+1} .

References

- 1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-Coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.
- 2 Vikraman Arvind, Abhranil Chatterjee, Rajit Datta, and Partha Mukhopadhyay. Fast Exact Algorithms Using Hadamard Product of Polynomials. *CoRR*, abs/1807.04496, 2018. arXiv:1807.04496.
- 3 Vikraman Arvind, Abhranil Chatterjee, Rajit Datta, and Partha Mukhopadhyay. Univariate Ideal Membership Parameterized by Rank, Degree, and Number of Generators. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, pages 7:1–7:18, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.7.
- 4 Garrett Birkhoff and Saunders Mac Lane. *Algebra*. AMS, 1999.
- 5 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: Fast subset convolution. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 67–74, 2007. doi:10.1145/1250790.1250801.
- 6 Andreas Björklund, Vikram Kamat, Lukasz Kowalik, and Meirav Zehavi. Spotting Trees with Few Leaves. *SIAM J. Discrete Math.*, 31(2):687–713, 2017. doi:10.1137/15M1048975.
- 7 Andreas Björklund, Petteri Kaski, and Ioannis Koutis. Directed Hamiltonicity and Out-Branchings via Generalized Laplacians. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 91:1–91:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.91.
- 8 Cornelius Brand. *Paths and Walks, Forests and Planes*. PhD thesis, Saarland University, 2019.
- 9 Cornelius Brand, Holger Dell, and Thore Husfeldt. Extensor-coding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 151–164, New York, NY, USA, 2018. ACM. doi:10.1145/3188745.3188902.

- 10 Jianer Chen, Songjian Lu, Sing-Hoi Sze, and Fenghui Zhang. Improved algorithms for path, matching, and packing problems. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 298–307, 2007. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283415>.
- 11 Nathann Cohen, Fedor V. Fomin, Gregory Z. Gutin, Eun Jung Kim, Saket Saurabh, and Anders Yeo. Algorithm for finding k-vertex out-trees and its application to k-internal out-branching problem. *J. Comput. Syst. Sci.*, 76(7):650–662, 2010. doi:10.1016/j.jcss.2010.01.001.
- 12 Artur Czumaj, editor. *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*. SIAM, 2018. doi:10.1137/1.9781611975031.
- 13 Jean Daligault. *Techniques combinatoires pour les algorithmes paramétrés et les noyaux, avec applications aux problèmes de multicoupe. (Combinatorial Techniques for Parameterized Algorithms and Kernels, with Applications to Multicut.)*. PhD thesis, Montpellier 2 University, France, 2011. URL: <https://tel.archives-ouvertes.fr/tel-00804206>.
- 14 Henning Fernau, Serge Gaspers, and Daniel Raible. Exact and parameterized algorithms for max internal spanning tree. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 100–111. Springer, 2009.
- 15 Uffe Flarup, Pascal Koiran, and Laurent Lyaudet. On the Expressive Power of Planar Perfect Matching and Permanents of Bounded Treewidth Matrices. In *Algorithms and Computation, 18th International Symposium, ISAAC 2007, Sendai, Japan, December 17-19, 2007, Proceedings*, pages 124–136, 2007. doi:10.1007/978-3-540-77120-3_13.
- 16 Fedor V Fomin, Serge Gaspers, Saket Saurabh, and Stéphan Thomassé. A linear vertex kernel for maximum internal spanning tree. *Journal of Computer and System Sciences*, 79(1):1–6, 2013.
- 17 Fedor V. Fomin, Fabrizio Grandoni, Daniel Lokshtanov, and Saket Saurabh. Sharp Separation and Applications to Exact and Parameterized Algorithms. *Algorithmica*, 63(3):692–706, 2012. doi:10.1007/s00453-011-9555-9.
- 18 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Representative Sets of Product Families. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 443–454. Springer, 2014. doi:10.1007/978-3-662-44777-2_37.
- 19 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.
- 20 Ira M. Gessel and Richard P. Stanley. Algebraic Enumeration. In R. L. Graham, M. Grötschel, and L. Lovász, editors, *Handbook of Combinatorics (Vol. 2)*, pages 1021–1061. MIT Press, Cambridge, MA, USA, 1995. URL: <http://dl.acm.org/citation.cfm?id=233228.233231>.
- 21 Gregory Gutin, Igor Razgon, and Eun Jung Kim. Minimum leaf out-branching and related problems. *Theoretical Computer Science*, 410(45):4571–4579, 2009.
- 22 Gregory Z. Gutin, Felix Reidl, Magnus Wahlström, and Meirav Zehavi. Designing deterministic polynomial-space algorithms by color-coding multivariate polynomials. *J. Comput. Syst. Sci.*, 95:69–85, 2018. doi:10.1016/j.jcss.2018.01.004.
- 23 Mark Jerrum and Marc Snir. Some Exact Complexity Results for Straight-Line Computations over Semirings. *J. ACM*, 29(3):874–897, 1982. doi:10.1145/322326.322341.
- 24 Ioannis Koutis. Faster Algebraic Algorithms for Path and Packing Problems. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfssdóttir, 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 575–586. Springer, 2008. doi:10.1007/978-3-540-70575-8_47.
- 25 Ioannis Koutis and Ryan Williams. Algebraic fingerprints for faster algorithms. *Commun. ACM*, 59(1):98–105, 2016. doi:10.1145/2742544.

- 26 Ioannis Koutis and Ryan Williams. Limits and Applications of Group Algebras for Parameterized Problems. *ACM T. Algorithms*, 12(3):31:1–31:18, 2016. doi:10.1145/2885499.
- 27 Wenjun Li, Yixin Cao, Jianer Chen, and Jianxin Wang. Deeper local search for parameterized and approximation algorithms for maximum internal spanning tree. *Inf. Comput.*, 252:187–200, 2017. doi:10.1016/j.ic.2016.11.003.
- 28 Kevin Pratt. Faster Algorithms via Waring Decompositions. *CoRR*, abs/1807.06194, 2018. arXiv:1807.06194.
- 29 Elena Prieto and Christian Sloper. Reducing to independent set structure: the case of k-internal spanning tree. *Nordic Journal of Computing*, 12(3):308–318, 2005.
- 30 Andrew James Stothers. *On the complexity of matrix multiplication*. PhD thesis, The University of Edinburgh, 2010.
- 31 Seinosuke Toda. Classes of arithmetic circuits capturing the complexity of computing the determinant. *IEICE Transactions on Information and Systems*, 75(1):116–124, 1992.
- 32 Chris Umans. Fast generalized DFTs for all finite groups. *CoRR*, abs/1901.02536, 2019. arXiv:1901.02536.
- 33 Ryan Williams. Finding paths of length k in $O(2^k)$ time. *Inform. Process. Lett.*, 109(6):315–318, 2009. doi:10.1016/j.ipl.2008.11.004.
- 34 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898, 2012. doi:10.1145/2213977.2214056.
- 35 Michał Włodarczyk. Clifford Algebras Meet Tree Decompositions. 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 29:1–29:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPICs.IPEC.2016.29.
- 36 Meirav Zehavi. Mixing Color Coding-Related Techniques. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 1037–1049, 2015. doi:10.1007/978-3-662-48350-3_86.
- 37 Meirav Zehavi. Mixing Color Coding-Related Techniques. In *Proceedings of the 23rd Annual European Symposium on Algorithms (ESA)*, volume 9294, pages 1037–1049. Springer, 2015. doi:10.1007/978-3-662-48350-3_86.
- 38 Meirav Zehavi. Algorithms for k-Internal Out-Branching and k-Tree in Bounded Degree Graphs. *Algorithmica*, 78(1):319–341, May 2017. doi:10.1007/s00453-016-0166-3.

On Geometric Set Cover for Orthants

Karl Bringmann

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

Sándor Kisfaludi-Bak

Department of Mathematics and Computer Science, Eindhoven University of Technology,
Eindhoven, The Netherlands
s.kisfaludi.bak@tue.nl

Michał Pilipczuk

Institute of Informatics, University of Warsaw, Poland
michal.pilipczuk@mimuw.edu.pl

Erik Jan van Leeuwen

Dept. Information & Computing Sciences, Utrecht University, Utrecht, The Netherlands
e.j.vanleeuwen@uu.nl

Abstract

We study SET COVER for orthants: Given a set of points in a d -dimensional Euclidean space and a set of orthants of the form $(-\infty, p_1] \times \dots \times (-\infty, p_d]$, select a minimum number of orthants so that every point is contained in at least one selected orthant. This problem draws its motivation from applications in multi-objective optimization problems. While for $d = 2$ the problem can be solved in polynomial time, for $d > 2$ no algorithm is known that avoids the enumeration of all size- k subsets of the input to test whether there is a set cover of size k . Our contribution is a precise understanding of the complexity of this problem in any dimension $d \geq 3$, when k is considered a parameter:

- For $d = 3$, we give an algorithm with runtime $n^{\mathcal{O}(\sqrt{k})}$, thus avoiding exhaustive enumeration.
- For $d = 3$, we prove a tight lower bound of $n^{\Omega(\sqrt{k})}$ (assuming ETH).
- For $d \geq 4$, we prove a tight lower bound of $n^{\Omega(k)}$ (assuming ETH).

Here n is the size of the set of points plus the size of the set of orthants. The first statement comes as a corollary of a more general result: an algorithm for SET COVER for half-spaces in dimension 3. In particular, we show that given a set of points U in \mathbb{R}^3 , a set of half-spaces \mathcal{D} in \mathbb{R}^3 , and an integer k , one can decide whether U can be covered by the union of at most k half-spaces from \mathcal{D} in time $|\mathcal{D}|^{\mathcal{O}(\sqrt{k})} \cdot |U|^{\mathcal{O}(1)}$.

We also study approximation for SET COVER for orthants. While in dimension 3 a PTAS can be inferred from existing results, we show that in dimension 4 and larger, there is no 1.05-approximation algorithm with runtime $f(k) \cdot n^{o(k)}$ for any computable f , where k is the optimum.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Theory of computation \rightarrow Computational complexity and cryptography

Keywords and phrases Set Cover, parameterized complexity, algorithms, Exponential Time Hypothesis

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.26

Funding *Sándor Kisfaludi-Bak*: Supported by the Netherlands Organization for Scientific Research NWO under project no. 024.002.003.

Michał Pilipczuk: This work is a part of project TOTAL (Mi. Pilipczuk) 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).



European Research Council



Acknowledgements We are grateful to an anonymous reviewer who pointed to us the reduction of Pach and Tardos [46] from ORTHANT COVER to GEOMETRIC SET COVER for half-spaces, and who suggested taking a closer look at the case of GEOMETRIC SET COVER for half-spaces. We would like to also thank the reviewer for pointing out that a PTAS for ORTHANT COVER in dimension 3 follows



© Karl Bringmann, Sándor Kisfaludi-Bak, Michał Pilipczuk, and Erik Jan van Leeuwen;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 26; pp. 26:1–26:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

from the work of Mustafa and Ray [44] composed with the reduction of Pach and Tardos [46], which replaced our previous ad-hoc argument. We thank the organizers of the workshop on fixed-parameter computational geometry, held in May 2018 at Lorentz Center in Leiden, the Netherlands, where the main conceptual part of this work was done.

1 Introduction

Motivated by applications in multi-objective optimization, we study a geometric variant of the classic SET COVER problem. In general, SET COVER is defined as follows. Let \mathbb{U} be a universe; typically, \mathbb{U} is a finite collection of elements or \mathbb{R}^d for some constant $d \geq 1$. Given a finite set $U \subseteq \mathbb{U}$ and a finite set \mathcal{T} of subsets of \mathbb{U} , the goal is to find a set $S \subseteq \mathcal{T}$ of minimum size such that for each $u \in U$ it holds that $u \in F$ for some $F \in S$. We let $n = |\mathcal{T}| + |U|$.

SET COVER can be approximated within factor $\ln |\mathcal{T}|$ by a greedy algorithm [13, 29, 34], but, unless $P = NP$, no polynomial-time algorithm can attain an approximation factor of $(1 - \varepsilon) \ln |\mathcal{T}|$ for any $\varepsilon > 0$ [20]. Moreover, when parameterized by the expected solution size k (formally, here we consider a decision problem), the problem is W[2]-hard [21] and there is no $\mathcal{O}(n^{k-\varepsilon})$ -time algorithm for any $\varepsilon > 0$, unless the Strong Exponential Time Hypothesis (SETH) fails [50]. Recently, it was even shown that, unless the Gap Exponential Time Hypothesis (Gap-ETH) fails, SET COVER has no $f(\text{OPT})|\mathcal{T}|^{o(\text{OPT})}$ -time algorithm that approximates the optimum OPT within a factor of $a(\text{OPT})$, for any computable a and f [9]. This makes SET COVER a very hard algorithmic problem in general.

Fortunately, through years of research, we know that SET COVER becomes much easier when geometry is involved. If the universe \mathbb{U} is equal to \mathbb{R}^d for some $d \geq 1$, the set U is a set of points, and the sets in \mathcal{T} are defined by geometric objects, then the problem is known as GEOMETRIC SET COVER. Then various restrictions on the shapes of objects allowed in \mathcal{T} may lead to different tractability results. While for $d = 1$ the problem is polynomial-time solvable when \mathcal{T} is required to consist of intervals, there are easy cases in $d = 2$ that are NP-hard, such as when \mathcal{T} is defined by sets of unit squares or disks [24, 30]. However, the approximability of GEOMETRIC SET COVER in $d = 2$ is significantly better than in general. Approaches like the shifting technique [23], ε -nets [1, 2, 8, 14, 33, 41, 47], local search [3, 25, 44], sampling techniques [11, 55] and separator techniques [43] have proven successful in obtaining constant-factor approximation algorithms and approximation schemes. Recently, Govindarajan et al. [25] showed a very general approximability result, namely that GEOMETRIC SET COVER admits a PTAS when the underlying sets in \mathcal{T} are non-piercing regions, which includes the case of pseudo-disks. From a parameterized perspective, Marx and Pilipczuk [39] showed that GEOMETRIC SET COVER has a $|\mathcal{T}|^{\mathcal{O}(\sqrt{k})}$ -time algorithm when \mathcal{T} is a set of disks or a set of squares. Moreover, no $n^{o(\sqrt{k})}$ -time algorithm exists for these cases unless the Exponential Time Hypothesis fails [36, 39]. For piercing regions, such as axis-parallel rectangles and fat triangles, GEOMETRIC SET COVER is APX-hard [10, 27] and admits no $|\mathcal{T}|^{o(k)}$ -time algorithm unless ETH fails [39]. For $d = 3$, a generic PTAS is also unlikely, as GEOMETRIC SET COVER is APX-hard even for unit balls [10], although constant-factor approximation algorithms do exist in certain cases [33]. This makes the complexity of GEOMETRIC SET COVER highly interesting for $d \geq 3$.

Orthant Cover. In this paper, we contribute to the knowledge about GEOMETRIC SET COVER by considering the case when the sets in \mathcal{T} are orthants, which we call ORTHANT COVER. An *orthant* is a subset $T \subset \mathbb{R}^d$ of the form $T = \{(x_1, \dots, x_d) \in \mathbb{R}^d : x_i \leq p_i \text{ for all } i \in [d]\}$ for some $(p_1, \dots, p_d) \in \mathbb{R}^d$. Alternatively, an orthant can be defined as $(-\infty, p_1] \times \dots \times (-\infty, p_d]$.

Our interest in ORTHANT COVER is motivated by multi-objective optimization. Here an optimization problem (like shortest path) is associated with $d > 1$ objectives (e.g. every edge has a cost and transition time), see, e.g., [5, 22, 26, 45, 48, 49]. We identify each possible solution of the optimization problem with the vector in \mathbb{R}^d that lists all of its d objectives. A solution $p \in \mathbb{R}^d$ is called *Pareto-optimal* if there is no solution $q \in \mathbb{R}^d$ with $q \geq p$ (i.e., $q_i \geq p_i$ for all $1 \leq i \leq d$). The set of all Pareto-optimal solutions $F \subseteq \mathbb{R}^d$ is called the *Pareto front* [32] (or *trade-off curve* [56] or *skyline* [4]), and computing it is the standard goal of multi-objective optimization.

However, the Pareto front can be prohibitively large to display to the end user. Therefore, a typical relaxation is to compute a $(1 + \varepsilon)$ -approximation of the Pareto front. This is defined as a subset F' of the Pareto front F such that for every $p \in F$ there exists a $q \in F'$ with $p \leq (1 + \varepsilon)q$ [48]. The question then becomes to find a Pareto front approximation of minimum size. This problem has been studied in multi-objective optimization under different names like “approximately dominating representatives” (ADR) [32] and “ ε -indicator subset selection” [6, 7, 57]. Observe that we can solve ADR using an algorithm for ORTHANT COVER by setting

$$U := F \quad \text{and} \quad \mathcal{T} := \{(-\infty, (1 + \varepsilon)f_1] \times \dots \times (-\infty, (1 + \varepsilon)f_d] : (f_1, \dots, f_d) \in F\}.$$

Therefore, ORTHANT COVER can be seen as an asymmetric variant of ADR. This provides strong motivation to gain an algorithmic understanding of ORTHANT COVER.

We already know that in dimension $d = 2$, ORTHANT COVER can be solved in polynomial time, and even in near-linear time in n [7, 32]. For $d \geq 3$, however, the problem becomes NP-hard [32]. Moreover, if we focus on looking for a solution of size at most k , no algorithm is known that avoids the enumeration of all size- k subsets of \mathcal{T} . In fact, no $n^{o(k)}$ -time algorithm is known, even for $d = 3$. Therefore, we ask *in which dimensions can the naive algorithm for ORTHANT COVER with running time $n^{O(k)}$ be significantly improved upon?*

Our Contribution. In this paper, we resolve the parameterized complexity of ORTHANT COVER when parameterized by the size of the solution. We present an algorithm for $d = 3$ that improves on the naive $n^{O(k)}$ -time algorithm, and rule out any further significant improvements in any dimension. Our lower bounds are conditional on the Exponential Time Hypothesis (ETH) by Impagliazzo, Paturi, and Zane [28], which (avoiding technical details) states that 3-SAT has no algorithm with running time $2^{o(n)}$, where n is the number of variables.

► **Theorem 1.** *Consider the ORTHANT COVER problem in dimension d . Then:*

1. *for $d = 3$, it can be solved in time $|\mathcal{T}|^{O(\sqrt{k})} \cdot |U|^{O(1)}$, in particular in time $n^{O(\sqrt{k})}$;*
2. *for $d = 3$, it cannot be solved in time $f(k)n^{o(\sqrt{k})}$ for any computable f , assuming ETH;*
3. *for $d \geq 4$, it cannot be solved in time $f(k)n^{o(k)}$ for any computable f , assuming ETH.*

In the above and for all the results stated in this paper, we measure the running time in the number of arithmetic operations over the reals given on input, i.e., in the strong fashion. Note that $n = |\mathcal{T}| + |U|$.

Thus, we determine the optimal time complexity of ORTHANT COVER as $n^{\Theta(\sqrt{k})}$ for $d = 3$ and $n^{\Theta(k)}$ for $d \geq 4$, assuming ETH. This dependence on d is somewhat surprising, since many previous conditional lower bounds for geometric problems are of the form $n^{\Omega(k^{1-1/d})}$ [40, 52]. We are only aware of one other work establishing problems to be easier for $d = 3$, but for $d = 4$ to be as hard as in any high dimension, namely k -means and k -median [15].

The algorithm of Theorem 1.1 actually follows from a more general result.

► **Theorem 2.** *Given a set of points U in \mathbb{R}^3 , a set of half-spaces \mathcal{D} in \mathbb{R}^3 , and an integer k , one can decide whether U can be covered by the union of at most k half-spaces from \mathcal{D} in time $|\mathcal{D}|^{\mathcal{O}(\sqrt{k})} \cdot |U|^{\mathcal{O}(1)}$.*

It is known that ORTHANT COVER can be reduced to this case (see [46, Lemma 2.3] or [12, Section A.3]). We observe that GEOMETRIC SET COVER for disks in \mathbb{R}^2 (DISK COVER) can also be reduced to this case, as follows. Consider an instance of DISK COVER where the point set U and the disk set \mathcal{T} are in the plane $z = 0$, and let p be a point in \mathbb{R}^3 outside this plane. For each disk $D \in \mathcal{T}$, we can define a ball $B(D)$ whose intersection with the plane $z = 0$ is D and that has p on its boundary. We apply an inversion with center p . As a result, each ball $B(D)$ is mapped to a half-space that contains the inverse of a point $x \in U$ if and only if x is covered by D . Hence, Theorem 2 also generalizes the known $n^{\mathcal{O}(\sqrt{k})}$ -time algorithm for DISK COVER [39].

We also study the approximability of ORTHANT COVER. Previous work implies a PTAS for $d = 3$ running in $n^{\mathcal{O}(1/\varepsilon^2)}$ time by a reduction (see [46, Lemma 2.3] or [12, Section A.3]) to the known PTAS for half-spaces in dimension 3 [44], and APX-hardness for $d \geq 4$ by a reduction (see Section 4) to the known APX-hardness of RECTANGLE COVER [54]. In this paper, we rule out any significant improvement for $d = 3$, particularly the existence of an Efficient PTAS. For $d \geq 4$, we establish a stronger inapproximability result conditional on Gap-ETH [19, 35].

► **Theorem 3.** *Consider the ORTHANT COVER problem in dimension d . Then:*

1. *for $d = 3$, it has no PTAS with running time $f(\varepsilon) n^{\mathcal{O}(\sqrt{1/\varepsilon})}$ for any computable f , assuming ETH;*
2. *for any $d \geq 4$, it has no 1.05-approximation algorithm running in time $f(k)n^{\mathcal{O}(k)}$ for any computable f , assuming Gap-ETH.*

Technical Overview. Our algorithm for half-spaces in \mathcal{R}^3 is a branching algorithm that attempts to split the input point set based on a balanced separator \mathcal{S}_0 of the optimum solution, where the separator should be small: of size $\mathcal{O}(\sqrt{k})$. However, we do not know the optimum solution and thus we cannot know the separator. Instead, we show that we can enumerate a set of candidate separators in time $|\mathcal{T}|^{\mathcal{O}(\sqrt{k})}$, in which the separator \mathcal{S}_0 is guaranteed to be contained. Similar approaches to obtain a subexponential-time algorithm for geometric and planar problems are known to exist (e.g. [31, 39]). However, the existence of the balanced separator of size $\mathcal{O}(\sqrt{k})$ is somewhat surprising here, since in 3 dimensions only separators of size $\mathcal{O}(k^{2/3})$ are known (see e.g. [53]). In order to get the desired separator size, we work on the surface of the convex polytope which is defined as the complement of the union of half-spaces in the solution. The edge graph of this polytope is planar, which allows us to define an appropriately small separator of the input point set.

For the $n^{\Omega(\sqrt{k})}$ lower bound, the first observation is that ORTHANT COVER for $d = 3$ is at least as hard as GEOMETRIC SET COVER in the plane where the objects are translates of an equilateral triangle. For the problem of GEOMETRIC SET COVER for squares, an $n^{\Omega(\sqrt{k})}$ conditional lower bound is known via a reduction from the GRID TILING problem [37, 40]. In this reduction, it is crucial that a gadget of (shifts of) a square can “transport” a value a from its left side to its right side, and a value b from its top side to its bottom side. For the related DOMINATING SET problem on intersection graphs of triangle translates, the proof strategy does generalize [18]. However, triangle translates are not flexible enough to naively follow this proof strategy for GEOMETRIC SET COVER: in a sense they have too few sides. Therefore, while our lower bound is also a reduction from GRID TILING, it is much more

subtle. Our most crucial construction is a “sumcheck” gadget that obtains “input values” a and b at two sides of the involved triangles and results in the value $a + b$ at the “output side”, while disallowing certain combinations. Using the “sumcheck” gadget on the values $a + b$ and $-a$ allows us to recover value b , and similarly we can recover a . Hence, we can use an essentially planar layout of triangles to transfer a value a from left to right and a value b from top to bottom; see Figure 3 for illustrations. Using this construction we can then simulate GRID TILING, obtaining the claimed lower bound.

The $n^{\Omega(k)}$ lower bound for $d = 4$ as well as our results on approximation algorithms all follow by relatively simple reductions to or from known results.

Organization. We prove Theorem 2 in Section 2, which implies Theorem 1.1 as per [46, Lemma 2.3] or [12, Section A.3]. We prove the remainder of Theorem 1 in Sections 3 and 4, which respectively contain a sketch of the lower bound for $d = 3$ and the lower bound for $d = 4$. Details of the lower bound for $d = 3$ as well as the proof of Theorem 3 are deferred to the full version of the paper.

2 Half-spaces in dimension 3

In this section, we prove Theorem 2 (and by extension, Theorem 1.1) by giving an algorithm for GEOMETRIC SET COVER for half-spaces in \mathbb{R}^3 . An instance of this problem consists of a set of half-spaces \mathcal{D} in \mathbb{R}^3 , a set of points U in \mathbb{R}^3 , and an integer k . The question is whether one can select k half-spaces from \mathcal{D} so that every point of U is covered by at least one of them.

We shall say that a set of half-spaces \mathcal{D} in \mathbb{R}^3 is *in general position* if no two boundaries of half-spaces in \mathcal{D} are parallel, and no four boundaries of half-spaces in \mathcal{D} meet at one point. Note that given an instance (\mathcal{D}, U, k) of GEOMETRIC SET COVER for half-spaces, one may slightly perturb the half-spaces of \mathcal{D} so that every half-space still covers the same subset of points in U as before, but after the perturbation they are in general position. Hence, we shall assume this property in all the considered instances of GEOMETRIC SET COVER for half-spaces.

2.1 Algorithm

Our algorithm will rely on the following balanced separator lemma.

► **Lemma 4.** *Suppose (\mathcal{D}, U, k) is an instance of GEOMETRIC SET COVER for half-spaces in \mathbb{R}^3 where \mathcal{D} is in general position, and let $\mathcal{S} \subseteq \mathcal{D}$ be an optimum solution to this instance, whose size ℓ satisfies $4 < \ell \leq k$. Then there exists a subset $\mathcal{S}_0 \subseteq \mathcal{S}$ with $|\mathcal{S}_0| \leq \mathcal{O}(\sqrt{k})$ and a partition \mathcal{P} of $U \setminus \bigcup \mathcal{S}_0$ with $|\mathcal{P}| \leq k$, such that the following property holds: For each $W \in \mathcal{P}$, if ℓ_W is the optimum size of a solution to the instance (\mathcal{D}, W, k) , then $\ell_W \leq \frac{2}{3}\ell$ and $|\mathcal{S}_0| + \sum_{W \in \mathcal{P}} \ell_W \leq \ell$.*

Moreover, given (\mathcal{D}, U, k) , one can in time $|\mathcal{D}|^{\mathcal{O}(\sqrt{k})} \cdot |U|^{\mathcal{O}(1)}$ enumerate a family \mathcal{N} consisting of at most $|\mathcal{D}|^{\mathcal{O}(\sqrt{k})}$ pairs $(\mathcal{S}_0, \mathcal{P})$ with $\mathcal{S}_0 \subseteq \mathcal{D}$, \mathcal{P} a partition of $U \setminus \bigcup \mathcal{S}_0$ with $|\mathcal{P}| \leq k$, and the guarantee that \mathcal{N} contains at least one pair satisfying the property above.

Before we give a proof of Lemma 4, we show how it can be used to construct an algorithm as promised in Theorem 2. The algorithm is presented below using pseudo-code as Algorithm `halfSpaceCoverDim3`.

Algorithm 1 Algorithm `halfSpaceCoverDim3`.

Input: Instance (\mathcal{D}, U, k) of GEOMETRIC SET COVER for half-spaces in \mathbb{R}^3 with \mathcal{D} in general position

Output: An optimum solution $\mathcal{S} \subseteq \mathcal{D}$ provided it has size $\leq k$, or \perp otherwise

```

 $\mathcal{S} \leftarrow \perp$ 
for each  $\mathcal{C} \subseteq \mathcal{D}$  with  $|\mathcal{C}| \leq \min(k, 4)$  do
    if  $U \subseteq \bigcup \mathcal{C}$  and  $|\mathcal{C}| < |\mathcal{S}|$  then // convention:  $|\perp| = \infty$ 
         $\mathcal{S} \leftarrow \mathcal{C}$ 
if  $k \leq 4$  then
    return  $\mathcal{S}$ 
 $\mathcal{N} \leftarrow$  family enumerated using the algorithm of Lemma 4 for  $(\mathcal{D}, U, k)$ 
for each  $(\mathcal{S}_0, \mathcal{P}) \in \mathcal{N}$  do
    for each  $W \in \mathcal{P}$  do
         $\mathcal{S}_W \leftarrow$  halfSpaceCoverDim3 $(\mathcal{D}, W, \lfloor 2k/3 \rfloor)$ 
         $\mathcal{C} \leftarrow \mathcal{S}_0 \cup \bigcup_{W \in \mathcal{P}} \mathcal{S}_W$  // convention:  $\perp \cup X = \perp$ 
        if  $|\mathcal{C}| \leq k$  and  $|\mathcal{C}| < |\mathcal{S}|$  then
             $\mathcal{S} \leftarrow \mathcal{C}$ 
return  $\mathcal{S}$ 

```

As argued, we may assume that \mathcal{D} is in general position. First, we look through all candidates \mathcal{C} for a solution with $|\mathcal{C}| \leq 4$. In case any such \mathcal{C} covering U is found, we store the smallest one as the optimum solution. Next, provided $k > 4$, we apply the algorithm of Lemma 4 to the instance (\mathcal{D}, U, k) and enumerate a suitable family of pairs \mathcal{N} . For each $(\mathcal{S}_0, \mathcal{P}) \in \mathcal{N}$ we apply the algorithm recursively to all instances $(\mathcal{D}, W, \lfloor \frac{2}{3}k \rfloor)$ for $W \in \mathcal{P}$, yielding solutions \mathcal{S}_W . We then consider $\mathcal{C} = \mathcal{S}_0 \cup \bigcup_{W \in \mathcal{P}} \mathcal{S}_W$ as a candidate solution, provided none of \mathcal{S}_W is equal to \perp . Finally, we output the smallest candidate solution of size at most k found.

The correctness of the algorithm follows immediately from Lemma 4. Indeed, if (\mathcal{D}, U, k) admits a solution of size at most 4, then an optimum solution will be found in the initial search. Otherwise, Lemma 4 ensures us that for some pair $(\mathcal{S}_0, \mathcal{P}) \in \mathcal{N}$, the recursive calls of the algorithm will find solutions \mathcal{S}_W for $W \in \mathcal{P}$ which together with \mathcal{S}_0 form an optimum solution to (\mathcal{D}, U, k) .

We are left with bounding the time complexity of the algorithm. Let $C > 0$ be such that the algorithm of Lemma 4 always returns a family \mathcal{N} satisfying $|\mathcal{N}| \leq |\mathcal{D}|^{C\sqrt{k}}$. Let $T[d, k]$ be the maximum number of leaves of the recursion tree produced by the algorithm when applied to an instance with $|\mathcal{D}| = d$ and parameter k . Then $T[d, k] = 1$ for $k \leq 4$, while for $k > 4$ we have the following recursive inequality:

$$T[d, k] \leq k \cdot d^{C\sqrt{k}} \cdot T[d, \lfloor 2k/3 \rfloor].$$

Here, factor $k \cdot d^{C\sqrt{k}}$ comes from the fact that for at most $d^{C\sqrt{k}}$ pairs $(\mathcal{S}_0, \mathcal{P}) \in \mathcal{N}$ we apply the algorithm recursively to $|\mathcal{P}| \leq k$ instances with parameter $\lfloor 2k/3 \rfloor$. Unraveling the recursion, we have

$$T[d, k] \leq k^{\log_{3/2} k} \cdot d^{C\sqrt{k} \cdot (1 + (2/3)^{1/2} + (2/3)^{2/2} + (2/3)^{3/2} + \dots)} = k^{\log_{3/2} k} \cdot d^{C'\sqrt{k}} = d^{\mathcal{O}(\sqrt{k})},$$

where $C' = C \cdot \frac{1}{1 - (2/3)^{1/2}}$.

We conclude that the recursion tree for an instance with $d = |\mathcal{D}|$ and parameter k has at most $d^{\mathcal{O}(\sqrt{k})}$ leaves, so it also has $d^{\mathcal{O}(\sqrt{k})}$ nodes. The internal computation for each node takes time $d^{\mathcal{O}(\sqrt{k})} \cdot |U|^{\mathcal{O}(1)}$, so the total running time of $d^{\mathcal{O}(\sqrt{k})} \cdot |U|^{\mathcal{O}(1)}$ follows.

2.2 Balanced separator lemma

We now move to the proof of Lemma 4, which spans the remainder of this section.

Since \mathcal{S} is an optimum solution to (\mathcal{D}, U, k) , we have that \mathcal{S} is *minimal* in the following sense: there is no $S \in \mathcal{S}$ such that $S \subseteq \bigcup_{T \in \mathcal{S} \setminus \{S\}} T$. It turns out that this minimality condition together with the assumption $|\mathcal{S}| > 4$ implies that \mathcal{S} cannot cover the whole space; this is implied by the following result.

► **Theorem 5** (Danzer et al. [17]). *If a set of half-spaces \mathcal{S} in \mathbb{R}^3 is minimal and $\bigcup \mathcal{S} = \mathbb{R}^3$, then $|\mathcal{S}| \leq 4$.*

For every half-space $S \in \mathcal{S}$ we may choose an affine function $\varphi_S: \mathbb{R}^3 \rightarrow \mathbb{R}$ so that

$$S = \{x \in \mathbb{R}^3: \varphi_S(x) \leq 0\}.$$

In particular, we set $\varphi_S(x) = \langle x - v_S, n_S \rangle$, where v_S is a point in the boundary of S , the vector n_S is the normal of the boundary plane of S pointing away from S , and $\langle \cdot, \cdot \rangle$ denotes the inner product in \mathbb{R}^3 . Let $\bar{S} = \{x \in \mathbb{R}^3: \varphi_S(x) \geq 0\}$; that is, \bar{S} is the closure of the complement of S . Then the complement of $\bigcup \mathcal{S}$ is the interior of the polytope P defined as follows:

$$P = \bigcap_{S \in \mathcal{S}} \bar{S} = \{x \in \mathbb{R}^3: \varphi_S(x) \geq 0 \text{ for all } S \in \mathcal{S}\}.$$

By Theorem 5 we infer that P is non-empty.

We shall also assume from now on that the polytope P is bounded. This can be achieved by adding to \mathcal{S} up to 6 *dummy* half-spaces of the form $\{(x_1, x_2, x_3) \in \mathbb{R}^3: x_i \geq M\}$ and $\{(x_1, x_2, x_3) \in \mathbb{R}^3: x_i \leq -M\}$ for $i = 1, 2, 3$ and some large M , so that none of the dummy half-spaces covers any point of U . These may be perturbed slightly so that \mathcal{S} remains in general position. As we will not use the optimality of \mathcal{S} from now on, this can be safely done at the cost of replacing ℓ with $\ell + 6$ in all asymptotic bounds. Note that we do ensure that minimality of \mathcal{S} is maintained, and thus possibly less than 6 dummy half-spaces are added.

Recall that we denote $|\mathcal{S}| = \ell$. Thus, P is a bounded convex polytope in \mathbb{R}^3 with ℓ faces, one for each half-space of \mathcal{S} (this follows by minimality). Since \mathcal{S} is in general position, at each vertex of P three faces meet. Let H be a graph whose vertices are the vertices of P and whose edges are the edges of P . Observe that the boundary of P – which consists of its faces – is homeomorphic to a sphere, so this homeomorphism shows that H admits a drawing in the sphere with ℓ faces. In the following, we identify faces of H with the faces of P . Since every face f of P is a polygon, the boundary of f is a simple cycle in H . Therefore, H is a simple 3-regular plane graph (i.e. without loops and multiple edges connecting the same pair of vertices) that is connected and bridgeless.

Let H' be the *radial graph* of H : the vertex set of H' consists of vertices and faces of H , and in H' a vertex u is adjacent to a face f if and only if u lies on the boundary of f . Note that H' is bipartite, with the vertices and faces of H being the bipartition. Also, H' admits an embedding into a sphere constructed from the embedding of H as follows: for every face f pick an arbitrary point $x_f \in f$ representing it, and connect x_f with all vertices u lying on f using pairwise non-crossing curves within f . Observe that every face of H' is a 4-cycle, induced by two faces of H and the endpoints of an edge shared by them. Since H is connected and bridgeless, a straightforward argument shows that H' is 2-connected. Since H is 3-regular, it follows that $3|V(H)| = 2|E(H)|$, so by Euler's formula for polyhedra ($|V(H)| - |E(H)| + \ell = 2$), we have that $|V(H)| = 2\ell - 4$. Consequently, $|V(H')| = 3\ell - 4$.

We may now apply the following Cycle Separator Theorem of Miller.

► **Theorem 6** ([42], with simplified formulation). *Let G be a 2-connected plane graph on n vertices and let d be the maximum length of a face in G . Suppose $\mu: V(G) \rightarrow [0, 1]$ is a weight function on the vertices of G such that $\mu(V(G)) = \sum_{v \in V(G)} \mu(v) = 1$. Then there exists a simple cycle C in G of length at most $2\sqrt{2\lfloor d/2 \rfloor n}$ such that if R_1 and R_2 are the (open) connected regions of the plane with C removed, then the vertices contained in R_1 have total weight at most $2/3$, and the same holds for R_2 .*

On the vertex set of H' define the following weight function: $\mu(f) = \frac{1}{\ell}$ for every face f of H , and $\mu(u) = 0$ for every vertex u of H . By Theorem 6, in H' there exists a simple cycle C of length at most $4\sqrt{|V(H')|} = 4\sqrt{3\ell - 4}$ such that every connected component of $H' - C$ contains at most $\frac{2}{3}\ell$ vertices that correspond to faces of H . Let

$$C = (z_1, f_1, z_2, f_2, \dots, z_q, f_q),$$

where $2q$ is the length of C (thus $q \leq 2\sqrt{3\ell - 4}$), z_1, \dots, z_q are consecutive vertices of H visited by C , and f_1, \dots, f_q are consecutive faces of H visited by C .

Let Q be a closed poly-line in \mathbb{R}^3 with vertices z_1, \dots, z_q , connected with straight line segments in this order (cyclically). Then the segment between z_i and z_{i+1} (with indices behaving cyclically modulo q) is entirely contained in the face f_i of P . Thus, Q is a curve contained in the boundary of P (denoted further ∂P), so it splits ∂P (which is homeomorphic to a sphere) into two regions, say A_1 and A_2 .

We now color the faces of P in three colors as follows:

- faces incident to any of the vertices z_1, z_2, \dots, z_q are colored green;
- remaining faces are colored red or blue, depending whether they are contained in A_1 or A_2 .

Note that since three faces meet at each vertex z_i , there are at most $4\sqrt{3\ell - 4}$ green faces: f_1, \dots, f_q and at most one additional face per each vertex z_i . Also, red faces do not share edges with blue faces, because all faces intersecting Q (even at one point) are colored green. We treat the above coloring of faces of P also as a coloring of all the points of ∂P . Here, points on edges of P are colored green if any face incident to the edge is colored green, and they are colored red or blue if all incident faces are red or blue, respectively.

Let $X = \text{conv}\{z_1, \dots, z_q\}$. The asserted properties of C immediately yield the following.

▷ **Claim 7.** There are at most $4\sqrt{3\ell - 4}$ green faces, at most $\frac{2}{3}\ell$ red faces, and at most $\frac{2}{3}\ell$ blue faces. No red face shares any edge with any blue face. Moreover, if x is any blue point on ∂P and y is any red point on ∂P , then the straight line segment with endpoints x and y intersects X .

As faces of P are in one-to-one correspondence with the half-spaces of \mathcal{S} , we may talk about red, green, and blue half-spaces of \mathcal{S} . We next observe that the separating properties of C carry over to the points of U .

▷ **Claim 8.** If a point $u \in U$ is simultaneously covered by a red half-space from \mathcal{S} and by a blue half-space from \mathcal{S} , then it is also covered by a green half-space from \mathcal{S} .

Proof. Let A and B be respectively the red and the blue half-space covering u , and let f_A and f_B be the faces of P that correspond to A and B , respectively. Pick any point $x_A \in f_A$ and $x_B \in f_B$ and let Π be a plane through u , x_A , and x_B . We may choose x_A and x_B so that Π does not contain any vertex of P . Then $P \cap \Pi$ is a nonempty convex polygon, whose sides are colored red, green, and blue so that no red side is adjacent to any blue side. Moreover, the side containing x_A is red, while the side containing x_B is blue. Call these sides s_A and s_B , respectively.

Call a side s of $P \cap \Pi$ *separating* if its extension to a line separates u from $P \cap \Pi$ in the plane Π . Since $P \cap \Pi$ is convex, separating sides form an interval on the perimeter of $P \cap \Pi$. Moreover, since A and B cover u , it follows that s_A and s_B are separating. As s_A is red and s_B is blue, from the two claims above we conclude that there exists a green side s of $P \cap \Pi$ that is also separating. Then the half-space corresponding to the face of P containing s is green and it covers u , as required. \triangleleft

By Claim 8, we may partition U into three subsets:

- *green points* of U that are covered by some green half-space from \mathcal{S} ;
- *red points* of U that are covered only by red half-spaces from \mathcal{S} ;
- *blue points* of U that are covered only by blue half-spaces from \mathcal{S} .

Denote the above sets by U_G, U_R, U_B , respectively. In the following claims, roughly speaking we show that X can be used to separate red points of U from blue points of U . Call two points $u, v \in U$ *separated* by X if the straight line segment with endpoints u and v intersects X .

▷ **Claim 9.** For all $u \in U_R$ and $v \in U_B$, we have that u and v are separated by X .

Proof. Let I be the straight line segment with endpoints u and v .

Suppose first that I does not intersect the polytope P . Since both I and P are convex, there exists an affine functional $\psi: \mathbb{R}^3 \rightarrow \mathbb{R}$ such that $\psi(u) < 0$, $\psi(v) < 0$, but $\psi(x) \geq 0$ for all $x \in P$. We may moreover choose ψ so that there exists a vertex w of P for which $\psi(w) = 0$. Let the faces of P incident to w be contained in the boundaries of half-spaces S_1, S_2, S_3 . Since ψ is nonnegative on P , it follows that ψ can be written as a nonnegative linear combination of $\varphi_{S_1}, \varphi_{S_2}, \varphi_{S_3}$. Then $\varphi_{S_i}(u) < 0$ holds for some index $i \in \{1, 2, 3\}$, and similarly condition $\varphi_{S_j}(v) < 0$ holds for some index $j \in \{1, 2, 3\}$. Thus S_i covers u and S_j covers v , so S_i is necessarily red and S_j is necessarily blue. However, the faces corresponding to S_i and S_j share an edge incident to the vertex w . This contradicts Claim 7.

Now we know that I indeed intersects P . Let $J = I \cap P$. Note that since $u \in U_R$, the endpoint of J closer to u has to be red, for the half-space corresponding to the face of P containing this endpoint covers u . Similarly, the endpoint of J closer to v has to be blue. We conclude that, by Claim 7, the segment J has to intersect X . \triangleleft

▷ **Claim 10.** Suppose $u, v \in U \setminus U_G$ are separated by X . Then there is no half-space in \mathcal{S} that would simultaneously cover both u and v .

Proof. Let I be the straight line segment with endpoints u and v , and let x be any point of $I \cap X$.

Suppose first that x lies on ∂P . Since $x \in X$ and all faces of P incident to z_1, \dots, z_q are colored green, it follows that x is green. Let S be any half-space of \mathcal{S} corresponding to a green face on which x lies. As $x \in I$, we conclude that S either covers u or v , which contradicts the assumption that $u, v \notin U_G$.

Suppose now that x lies in the interior of P . If there was a half-space $S \in \mathcal{S}$ containing both u and v , then S would contain the whole segment I , and x in particular, so S would intersect the interior of P . This is a contradiction with the definition of P . \triangleleft

Consider now a graph L with vertex set $U \setminus U_G$, where different $u, v \in U \setminus U_G$ are considered adjacent if and only if they are not separated by X . Then Claims 9 and 10 directly imply the following.

▷ **Claim 11.** Every connected component of L is entirely contained either in U_R or in U_B . Moreover, no half-space in \mathcal{S} covers points from two different connected components of L .

26:10 On Geometric Set Cover for Orthants

The existential part of Lemma 4 follows now if we take \mathcal{S}_0 to be the set of green half-spaces and \mathcal{P} to be the partition of $U \setminus U_G$ into the connected components of L . Here, if any half-space from \mathcal{S}_0 turns out to be one of the at most six dummy half-spaces, we may safely remove it from \mathcal{S}_0 , as it does not cover any point in U anyway. Let us check that the required properties are indeed satisfied by the pair $(\mathcal{S}_0, \mathcal{P})$. First, by Claim 7 we have $|\mathcal{S}_0| \leq 2q \leq \mathcal{O}(\sqrt{k})$. Next, for a connected component W of L , let us denote by \mathcal{S}_W the set of half-spaces from \mathcal{S} that cover at least one point of $U \setminus U_G$ belonging to W . Clearly, \mathcal{S}_W is a solution to the instance (\mathcal{D}, W, k) , hence $|\mathcal{S}_W| \geq \ell_W$. By Claim 10, the sets \mathcal{S}_W are pairwise disjoint, and they are clearly disjoint from \mathcal{S}_0 . Hence, we have

$$\ell = |\mathcal{S}| \geq |\mathcal{S}_0| + \sum_{W \in \text{cc}(L)} |\mathcal{S}_W| \geq |\mathcal{S}_0| + \sum_{W \in \text{cc}(L)} \ell_W,$$

where $\text{cc}(L)$ is the set of connected components of L . Also, the sets \mathcal{S}_W are non-empty, because every connected component of L requires at least one half-space to be covered, so $\sum_{W \in \text{cc}(L)} |\mathcal{S}_W| \leq \ell$ entails that $|\mathcal{P}| = |\text{cc}(L)| \leq \ell \leq k$. Finally, by Claim 9, for each $W \in \text{cc}(L)$ the half-spaces of \mathcal{S}_W are either all red or all blue, which by Claim 7 implies that $|\mathcal{S}_W| \leq \frac{2}{3}\ell$ for all $W \in \text{cc}(L)$.

We are left with providing an algorithm enumerating a suitable family \mathcal{N} . The algorithm proceeds as follows. Let \mathcal{D}' be \mathcal{D} augmented by adding the six dummy half-spaces of the form $\{(x_1, x_2, x_3) \in \mathbb{R}^3 : x_i \geq M\}$ and $\{(x_1, x_2, x_3) \in \mathbb{R}^3 : x_i \leq -M\}$ for $i = 1, 2, 3$ and some large M , so that none of the added half-spaces covers any point of U . Say that a point $x \in \mathbb{R}^3$ is *important* if it is the intersection of some triple of planes that are boundaries of some half-spaces in \mathcal{D}' . Note that all vertices of the polytope P are important points, while the total number of important points is at most $(|\mathcal{D}| + 6)^3$ and they can be enumerated in time $\mathcal{O}(|\mathcal{D}|^3)$.

Next, for every $q \leq 2\sqrt{3 \cdot (k+6)} - 4 = 2\sqrt{3k+14}$, iterate through

- every choice of $2q$ half-spaces from \mathcal{D} , say $\mathcal{S}_0 = \{S_1, \dots, S_{2q}\}$;
- and every choice of q important points z_1, \dots, z_q .

Note that there are at most $|\mathcal{D}|^{4\sqrt{3k+14}}$ choices for \mathcal{S}_0 and at most $(|\mathcal{D}| + 6)^{6\sqrt{3k+14}}$ choices for z_1, \dots, z_q , hence we iterate through at most $(|\mathcal{D}| + 6)^{10\sqrt{3k+14}}$ choices in total.

Let $X = \text{conv}\{z_1, \dots, z_q\}$ and let U_G be the set of all points of U that are covered by some half-space of \mathcal{S}_0 . Construct the graph L as described before: the vertex set of L is $U \setminus U_G = U \setminus \bigcup \mathcal{S}_0$, and two points $u, v \in U \setminus U_G$ are adjacent if and only if u and v are not separated by X . Observe that whether u and v are separated by X can be checked in strongly polynomial time. Indeed, this question boils down to the verifying whether, in 3-dimensional Euclidean space, a given segment intersects a polyhedron defined as the convex hull of a given set of points, which can be solved by any strongly polynomial-time procedure for intersecting two convex polyhedra in \mathbb{R}^3 , see e.g. [51, Section 7.3 and notes and comments to Chapter 7]. Therefore, L can be computed in (strongly) polynomial time. Finally, if L has at most k connected components, then include in the constructed family \mathcal{N} the pair $(\mathcal{S}_0, \mathcal{P})$, where \mathcal{P} is the partition of $U \setminus U_G$ into the connected components of L .

The bound on the size of \mathcal{N} and the running time of the algorithm follow immediately from the description. The correctness is also clear, as some choice of \mathcal{S}_0 and z_1, \dots, z_q considered by the algorithm is the same as the one considered in the existential argument. This finishes the proof of Lemma 4.

3 Lower bound for dimension 3

The goal of this section is to prove Theorem 1, assertion 2. We can restrict our attention to the case where all points lie on the plane $\Pi = \{(x, y, z) : x + y + z = 0\}$ and the corners of all orthants lie on the plane $\{(x, y, z) : x + y + z = 1\}$. In such a setting, the intersections of the orthants with Π form equilateral triangles on Π , which all have the same size and orientation. In essence, this setting of ORTHANT COVER is equivalent to finding a geometric set cover of size k among m translates of some triangle. We call this problem TRIANGLE TRANSLATE COVER. Therefore, Theorem 1.2 is implied by the following theorem.

► **Theorem 12.** *There is no $f(k)n^{o(\sqrt{k})}$ algorithm for TRIANGLE TRANSLATE COVER for any computable function f , unless ETH fails.*

Here n is m plus the number of points.

Our reduction is from GRID TILING [38, 16], which is defined as follows. We are given as input an integer k , an integer n , and a collection \mathcal{S} of k^2 non-empty sets $S_{i,j} \subseteq \{1, \dots, n\} \times \{1, \dots, n\}$ for $i, j \in \{1, \dots, k\}$. The goal is to select an element $s_{i,j} \in S_{i,j}$ for each $i, j \in \{1, \dots, k\}$ such that:

- If $i < k$, $s_{i,j} = (x, y)$, and $s_{i+1,j} = (x', y')$, then $x = x'$.
- If $j < k$, $s_{i,j} = (x, y)$, and $s_{i,j+1} = (x', y')$, then $y = y'$.

One can picture these sets in a $k \times k$ matrix: in each cell (a, b) , we need to select a representative from the set $S_{i,j}$ so that the representatives selected from horizontally neighboring cells agree in the first coordinate, and representatives from vertically neighboring cells agree in the second coordinate. Observe that due to equality conditions, the goal in the GRID TILING problem can be stated equivalently as follows: select elements $x_1, \dots, x_k, y_1, \dots, y_k \in [n]$ such that $(x_i, y_j) \in S_{i,j}$ for all $i, j \in [k]$. Note that $s_{i,j} = (x_i, y_j)$ in this case. In the following, we will treat the selection $x_1, \dots, x_k, y_1, \dots, y_k$ also as a *solution* to a GRID TILING instance.

Our goal is to create a parameterized reduction where the constructed instance of TRIANGLE TRANSLATE COVER has a cover of size ck^2 for some constant c if and only if the original GRID TILING instance has a solution. This is sufficient due to the following theorem.

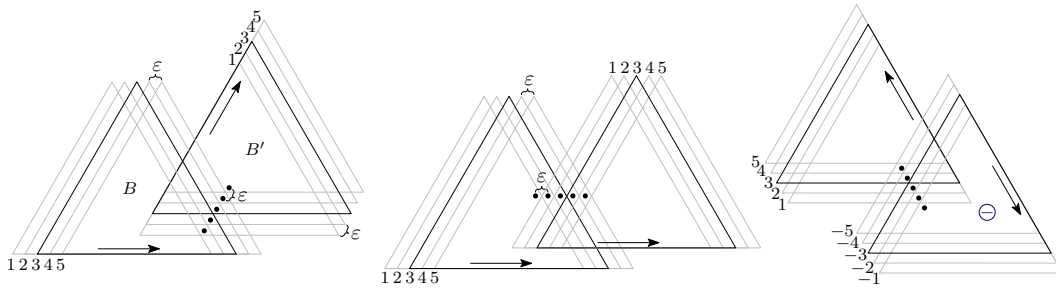
► **Theorem 13** ([38, 16]). *There is no $f(k)n^{o(k)}$ algorithm for GRID TILING for any computable function f , unless ETH fails.*

3.1 Gadgets

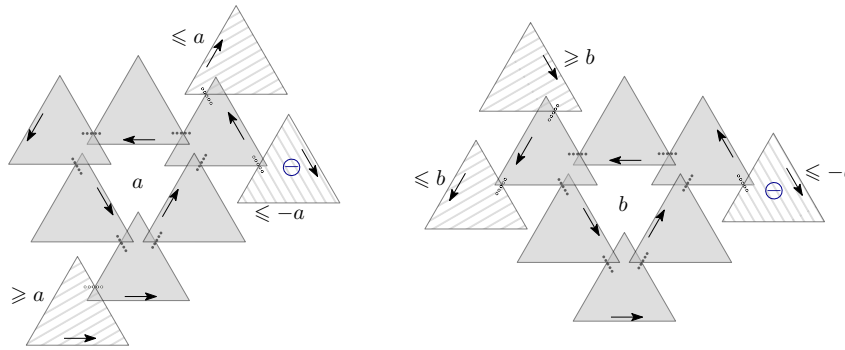
Due to lack of space, we only give a short intuitive overview of our construction. The complete construction and all proofs can be found in the full version of the paper.

Let $\varepsilon = \frac{1}{100n}$. In the TRIANGLE TRANSLATE COVER problem, the input triangles are equilateral triangles; we assume the side lengths are precisely 1. This means our construction can effectively use three directions, namely along the vectors $\bar{\mathbf{e}} = (1, 0)$, $\acute{\mathbf{e}} = (1/2, \sqrt{3}/2)$, and $\grave{\mathbf{e}} = (1/2, -\sqrt{3}/2)$. For convenience, we let $E = \{\bar{\mathbf{e}}, -\bar{\mathbf{e}}, \acute{\mathbf{e}}, -\acute{\mathbf{e}}, \grave{\mathbf{e}}, -\grave{\mathbf{e}}\}$. Given a positive integer N , we use $[N]$ to denote $\{1, \dots, N\}$ and $[-N]$ to denote $\{-1, \dots, -N\}$.

Bundles. We first establish a gadget to represent an integer value. Let $N = 2n$ and $\mathbf{e} \in E$. A *bundle* is a set of N triangles $B = \{t_1, \dots, t_N\}$ such that t_1 has its lower-left corner on the origin and t_{i+1} is t_i translated by $i\varepsilon \cdot \mathbf{e}$. The bundle also contains a point p_B on the incenter of $t_{N/2}$ ensuring that at least one triangle is selected from B . The idea behind the construction is that each solution will select exactly one triangle from the bundle. In this manner, the index of the selected triangle represents an integer in $[N]$. In the figures, each bundle has an arrow that indicates the direction along which the translation is done, and the indices (i.e. the represented integer) increase.



■ **Figure 1** Left, middle: transportation gadgets. Arrows indicate the direction of increasing indices within the bundle. Right: negation gadget.

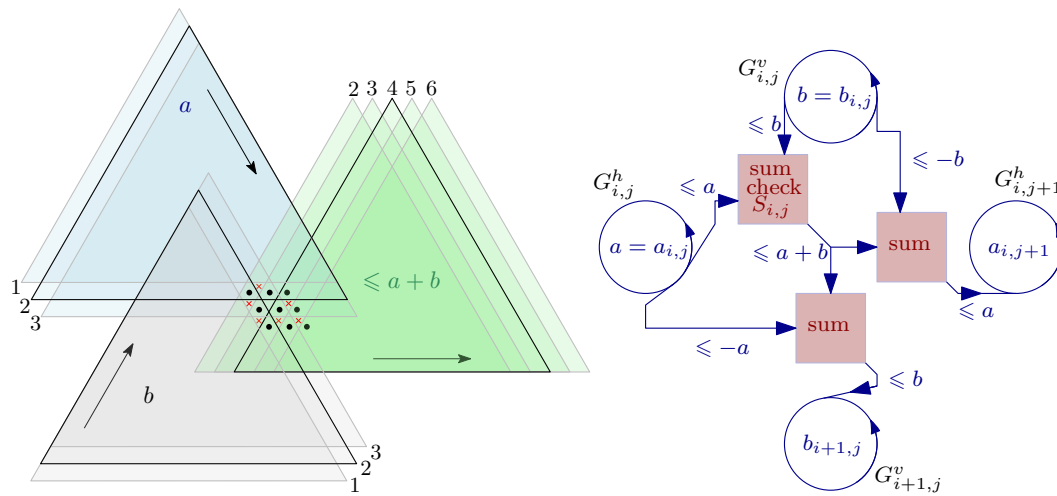


■ **Figure 2** Integer gadgets with some transportation to the outside.

A *negative bundle* uses a different indexing, and represents the integer $-N + i - 1$. Hence, the index of the selected triangle represents an integer in $[-N]$. (In figures, such bundles are indicated with a minus sign.)

Transportation gadget. We now establish a gadget to transport an integer value over some distance; this is built on a pair of bundles B and B' as in Figure 1. Observe that the boundaries of the triangles of B and B' induce an $(N - 1) \times (N - 1)$ lattice with directions \bar{e} and \bar{e} . Now place points in the cells of this lattice as indicated in the figure. In this way, we are able to transport the integer value i represented by the triangle selected from B to an integer value of at most i for the bundle B' . Note that within certain limits, we can translate B' at will, so that we can “lengthen” or “shorten” as needed for the rest of the construction. By switching the sign of the values represented by one bundle of the transportation gadget, we get a *negation gadget* (see right hand side of Figure 1); if we make a cycle by joining transportation gadgets, we get an *integer gadget*, within which the selected triangles of each bundle must represent the same integer value (see Figure 2).

Sum and sumcheck gadget. We can create a *sum gadget*, which has three bundles, two of which are considered input bundles and one an output bundle. The gadget has the property that if the triangles selected in the two input bundles represent a and b respectively, then the output bundle must have a triangle representing some value that is at most $a + b$. Such a gadget is depicted in Figure 3. By adding extra points (indicated by red crosses), we can also disable the selection of certain triplets $(a, b, a + b)$. For a set $S \subseteq [n] \times [n]$, this allows us to create a *sumcheck gadget* where given inputs a, b the output is at most $a + b$, where equality can occur if and only if $(a, b) \in S$. This is the crucial step that eventually allows us to check for the sets $S_{i,j}$ corresponding to the cell i, j of the GRID TILING instance.



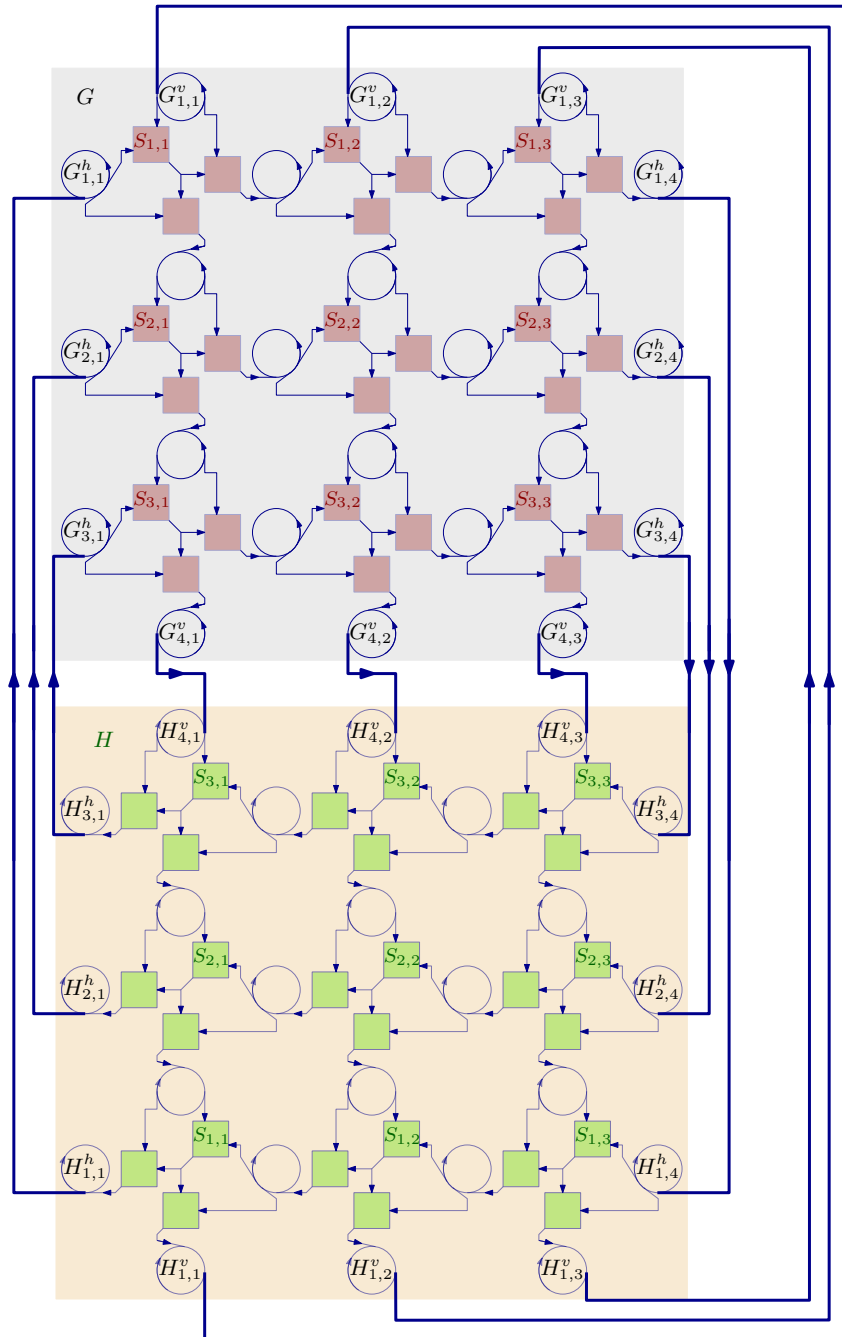
■ **Figure 3** Left: summation gadget. Adding the points indicated by the red crosses creates an S -sumcheck gadget for $S = \{(1, 1); (1, 3); (2, 2)\}$. Right: schematic representation of a cell (i, j) of the GRID TILING instance. The transportation gadgets (blue) carry inequalities involving $a = a_{i,j}$ and $b = b_{i,j}$.

3.2 The complete construction

The idea of the reduction is to have for any pair of neighboring tiles an integer gadget which contains the value that these neighboring tiles must agree on. Given such integer gadgets, we can realize a single tile $(i, j) \in [k] \times [k]$ using our gadgetry the following way. We want to transfer the value a that is our horizontal selection and the value b which is our vertical selection between these integer gadgets. At the same time, we want to ensure that $(a, b) \in S_{i,j}$. We do this as explained schematically in Figure 3. From the integer gadgets on the left and on the top, we extract the integer values stored there, say $a_{i,j}$ and $b_{i,j}$ respectively, and transport these values (using transportation gadgets) to an $S_{i,j}$ -sumcheck gadget. The output of this gadget will be an integer c satisfying $c \leq a_{i,j} + b_{i,j}$, and moreover $c < a_{i,j} + b_{i,j}$ if $(a_{i,j}, b_{i,j}) \notin S_{i,j}$. Using negation gadgets, we can extract $-a_{i,j}$ and $-b_{i,j}$ from the left and top integer gadgets, respectively. Each of these values can be combined with c through a sum gadget, whose output (i.e., third bundle) recovers integer values $a_{i,j+1} \leq a_{i,j}$ and $b_{i+1,j} \leq b_{i,j}$ that can be passed along to the right hand side and bottom integer gadgets respectively.

Let G be the construction thus far. Note that the construction ensures that left-to-right and top-to-bottom we have non-increasing values stored in our integer gadgets. To ensure equality holds, we need to wrap the rows and columns into cycles, just as we did for a single integer gadget. Doing this in a naive manner would lead to further crossings, so instead we create a construction H that is similar to G , but the rows are in reverse order, and the gadgetry of every tile is mirrored on the vertical axis; this construction is then translated below G (see Figure 4). In particular, the cell in row i and column j of H corresponds to the cell in row $k - i + 1$ and column j of the GRID TILING instance. As Figure 4 indicates, we can create transportation gadgets in a suitable manner to realize this construction.

We remark that it is tempting to use a known variant of GRID TILING called GRID TILING WITH \leq as the starting point of the reduction, which enjoys the same complexity lower bound as GRID TILING; see [16, Theorem 14.30]. In this variant, the equality conditions are replaced with the requirement that one coordinate behaves non-decreasingly along rows, while the second behaves non-decreasingly along columns. The variant looks convenient, as our



■ **Figure 4** Schematic representation of the complete construction for $k = 3$.

gadgets directly implement inequalities between coordinates, not equalities. This thinking, however, is problematic for the following reason: in our construction, in order to implement the check $(a, b) \in S_{i,j}$ we have to enforce equality in the sumcheck gadget created for the cell (i, j) , as in case of any slackness, this condition is not checked by the gadget. Therefore, starting the reduction from GRID TILING WITH \leq would not simplify the reasoning.

4 Lower bound for dimension 4 and higher

Consider the RECTANGLE COVER problem: Given points $P \subseteq \mathbb{R}^2$, a set R of axis-parallel rectangles in the plane, and a number k , decide whether there is a subset $R' \subseteq R$ of size k such that P is contained in the union of all rectangles in R' . RECTANGLE COVER is not solvable in time $f(k)n^{o(k)}$ for any computable f assuming ETH [39], where $n = |P| + |R|$. We obtain the same lower bound for ORTHANT COVER in dimension $d \geq 4$ by an easy reduction.

Proof of Theorem 1.3. Given points P and rectangles R in the plane, we construct a 4-dimensional ORTHANT COVER instance (U, \mathcal{T}) : For each point $p = (x, y) \in P$, we add the point $(-x, x, -y, y)$ to U . For each rectangle $r = [x_1, x_2] \times [y_1, y_2] \in R$, we add the orthant with corner $(-x_1, x_2, -y_1, y_2)$ to \mathcal{T} . Note that p is contained in r if and only if $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2$, which is equivalent to $-x \leq -x_1$, $x \leq x_2$, $-y \leq -y_1$, and $y \leq y_2$. For points $p, q \in \mathbb{R}^d$, note that q is contained in the orthant $T = (-\infty, p_1] \times \dots \times (-\infty, p_d]$ if and only if every coordinate of p is not larger than the corresponding coordinate of q . This proves the correctness of our reduction. We thus ruled out time $f(k)n^{o(k)}$ assuming ETH for ORTHANT COVER in dimension $d = 4$, and also for any $d \geq 4$ (by a trivial embedding). ◀

Together with the reasoning presented in [39], the above argument yields a chain of reductions from CLIQUE to ORTHANT COVER in $d = 4$. Using recent hardness of approximation for CLIQUE [9] and carefully tracking the gap through this chain of reductions, we obtain that ORTHANT COVER in $d = 4$ has no 1.05-approximation with running time $f(k)n^{o(k)}$ for any computable f , assuming Gap-ETH (Theorem 3.2). This result is presented in the full version of the paper.

References

- 1 Pankaj K. Agarwal and Jiangwei Pan. Near-Linear Algorithms for Geometric Hitting Sets and Set Covers. In Siu-Wing Cheng and Olivier Devillers, editors, *30th Annual Symposium on Computational Geometry, SOCG'14, Kyoto, Japan, June 08 - 11, 2014*, page 271. ACM, 2014.
- 2 Boris Aronov, Esther Ezra, and Micha Sharir. Small-Size ε -Nets for Axis-Parallel Rectangles and Boxes. *SIAM Journal on Computing*, 39(7):3248–3282, 2010.
- 3 Rom Aschner, Matthew J. Katz, Gila Morgenstern, and Yelena Yuditsky. Approximation Schemes for Covering and Packing. In Subir Kumar Ghosh and Takeshi Tokuyama, editors, *WALCOM: Algorithms and Computation, 7th International Workshop, WALCOM 2013, Kharagpur, India, February 14-16, 2013. Proceedings*, volume 7748 of *Lecture Notes in Computer Science*, pages 89–100. Springer, 2013.
- 4 Wolf-Tilo Balke, Ulrich Güntzer, and Jason Xin Zheng. Efficient Distributed Skylining for Web Information Systems. In *Advances in Database Technology, EDBT'04, 9th International Conference on Extending Database Technology*, volume 2992 of *LNCS*, pages 256–273. Springer, 2004.
- 5 René Beier and Berthold Vöcking. Random knapsack in expected polynomial time. *J. Comput. Syst. Sci.*, 69(3):306–329, 2004.

- 6 Karl Bringmann, Tobias Friedrich, and Patrick Klitzke. Generic Postprocessing via Subset Selection for Hypervolume and Epsilon-Indicator. In *13th International Conference on Parallel Problem Solving from Nature, PPSN XIII*, volume 8672 of *LNCS*, pages 518–527. Springer, 2014.
- 7 Karl Bringmann, Tobias Friedrich, and Patrick Klitzke. Two-dimensional subset selection for hypervolume and epsilon-indicator. In *Genetic and Evolutionary Computation Conference, GECCO'14*, pages 589–596. ACM, 2014.
- 8 Hervé Brönnimann and Michael T. Goodrich. Almost Optimal Set Covers in Finite VC-Dimension. *Discrete and Computational Geometry*, 14(1):463–479, 1995. doi:10.1007/BF02570718.
- 9 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 *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS'17*, pages 743–754, 2017.
- 10 Timothy M. Chan and Elyot Grant. Exact algorithms and APX-hardness results for geometric packing and covering problems. *Comput. Geom.*, 47(2):112–124, 2014.
- 11 Timothy M. Chan, Elyot Grant, Jochen Könemann, and Malcolm Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1576–1585. SIAM, 2012.
- 12 Timothy M. Chan, Kasper Green Larsen, and Mihai Patrascu. Orthogonal Range Searching on the RAM, Revisited. *CoRR*, abs/1103.5510, 2011. arXiv:1103.5510.
- 13 Vasek Chvátal. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4(3):233–235, 1979. doi:10.1287/moor.4.3.233.
- 14 Kenneth L. Clarkson and Kasturi R. Varadarajan. Improved Approximation Algorithms for Geometric Set Cover. *Discrete & Computational Geometry*, 37(1):43–58, 2007.
- 15 Vincent Cohen-Addad, Arnaud de Mesmay, Eva Rotenberg, and Alan Roytman. The Bane of Low-Dimensionality Clustering. In *29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '18*, pages 441–456. SIAM, 2018.
- 16 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 17 Ludwig Danzer, Branko Grünbaum, and Viktor Klee. Helly's theorem and its relatives. In *Proceedings of Symposia in Pure Mathematics*, volume 7, pages 101–180, 1963.
- 18 Mark de Berg, Sándor Kisfaludi-Bak, and Gerhard Woeginger. The complexity of Dominating Set in geometric intersection graphs. *Theoretical Computer Science*, 769:18–31, 2019. doi:10.1016/j.tcs.2018.10.007.
- 19 Irit Dinur. Mildly exponential reduction from gap 3SAT to polynomial-gap label-cover. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:128, 2016. URL: <http://eccc.hpi-web.de/report/2016/128>.
- 20 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 624–633. ACM, 2014.
- 21 Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness. *Congressus Numerantium*, 87:161–178, 1992.
- 22 Thomas Erlebach, Hans Kellerer, and Ulrich Pferschy. Approximating Multiobjective Knapsack Problems. *Management Science*, 48(12):1603–1612, 2002.
- 23 Thomas Erlebach and Erik Jan van Leeuwen. PTAS for weighted set cover on unit squares. In Maria J. Serna, Ronen Shaltiel, Klaus Jansen, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010, Barcelona, Spain, September 1-3, 2010. Proceedings*, volume 6302 of *Lecture Notes in Computer Science*, pages 166–177. Springer, 2010.

- 24 Robert J. Fowler, Mike S. Paterson, and Steven L. Tanimoto. Optimal Packing and Covering in the Plane are NP-Complete. *Information Processing Letters*, 12(3):133–137, 1981. doi:10.1016/0020-0190(81)90111-3.
- 25 Sathish Govindarajan, Rajiv Raman, Saurabh Ray, and Aniket Basu Roy. Packing and Covering with Non-Piercing Regions. In Piotr Sankowski and Christos Zaroliagis, editors, *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 47:1–47:17. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.
- 26 Pierre Hansen. Bicriterion path problems. In *Multiple Criteria Decision Making Theory and Application*, pages 109–127. Springer, 1980.
- 27 Sarel Har-Peled. Being Fat and Friendly is Not Enough. *CoRR*, abs/0908.2369, 2009. arXiv:0908.2369.
- 28 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.
- 29 David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, December 1974. doi:10.1016/S0022-0000(74)80044-9.
- 30 David S. Johnson. The NP-Completeness Column: An Ongoing Guide. *Journal of Algorithms*, 3(2):182–195, 1982. doi:10.1016/0196-6774(82)90018-9.
- 31 Sándor Kisfaludi-Bak, Jesper Nederlof, and Erik Jan van Leeuwen. Nearly ETH-tight algorithms for Planar Steiner Tree with terminals on Few Faces. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA’19*, 2019. to appear.
- 32 Vladlen Koltun and Christos H. Papadimitriou. Approximately dominating representatives. *Theoretical Computer Science*, 371(3):148–154, 2007.
- 33 Sören Laue. Geometric Set Cover and Hitting Sets for Polytopes in \mathbb{R}^3 . In Susanne Albers and Pascal Weil, editors, *STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings*, volume 08001 of *Dagstuhl Seminar Series*, pages 479–490. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2008. URL: <http://drops.dagstuhl.de/opus/volltexte/2008/1367>.
- 34 Lászlo Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13(4):383–390, 1975. doi:10.1016/0012-365X(75)90058-8.
- 35 Pasin Manurangsi and Prasad Raghavendra. A Birthday Repetition Theorem and Complexity of Approximating Dense CSPs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP’17*, volume 80 of *LIPIcs*, pages 78:1–78:15, 2017.
- 36 Dániel Marx. Parameterized Complexity of Independence and Domination on Geometric Graphs. In *Parameterized and Exact Computation, Second International Workshop, IWPEC, Proceedings*, pages 154–165, 2006. doi:10.1007/11847250_14.
- 37 Dániel Marx. On the Optimality of Planar and Geometric Approximation Schemes. In *48th Annual IEEE Symposium on Foundations of Computer Science, FOCS’07*, pages 338–348, 2007.
- 38 Dániel Marx. A Tight Lower Bound for Planar Multiway Cut with Fixed Number of Terminals. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 677–688. Springer, 2012.
- 39 Dániel Marx and Michał Pilipczuk. Optimal Parameterized Algorithms for Planar Facility Location Problems Using Voronoi Diagrams. In *ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 865–877. Springer, 2015. See arXiv:1504.05476 for the full version.
- 40 Dániel Marx and Anastasios Sidiropoulos. The limited blessing of low dimensionality: when $1 - 1/d$ is the best possible exponent for d -dimensional geometric problems. In *30th Annual Symposium on Computational Geometry, SOCG’14*, page 67. ACM, 2014.
- 41 Jiří Matoušek, Raimund Seidel, and Emo Welzl. How to Net a Lot with Little: Small epsilon-Nets for Disks and Halfspaces. In Raimund Seidel, editor, *Proceedings of the Sixth Annual Symposium on Computational Geometry, Berkeley, CA, USA, June 6-8, 1990*, pages 16–22. ACM, 1990.

- 42 Gary L. Miller. Finding Small Simple Cycle Separators for 2-Connected Planar Graphs. *J. Comput. Syst. Sci.*, 32(3):265–279, 1986.
- 43 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.
- 44 Nabil H. Mustafa and Saurabh Ray. Improved Results on Geometric Hitting Set Problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010. doi:10.1007/s00454-010-9285-9.
- 45 Frank Neumann. Expected runtimes of a simple evolutionary algorithm for the multi-objective minimum spanning tree problem. *European Journal of Operational Research*, 181(3):1620–1629, 2007.
- 46 János Pach and Gábor Tardos. Tight lower bounds for the size of epsilon-nets. In *Symposium on Computational Geometry*, pages 458–463. ACM, 2011.
- 47 János Pach and Gerhard J. Woeginger. Some New Bounds for Epsilon-Nets. In Raimund Seidel, editor, *Proceedings of the Sixth Annual Symposium on Computational Geometry, Berkeley, CA, USA, June 6-8, 1990*, pages 10–15. ACM, 1990.
- 48 Christos H. Papadimitriou and Mihalis Yannakakis. On the Approximability of Trade-offs and Optimal Access of Web Sources. In *41st Annual Symposium on Foundations of Computer Science, FOCS'00*, pages 86–92. IEEE Computer Society, 2000.
- 49 Christos H. Papadimitriou and Mihalis Yannakakis. Multiobjective Query Optimization. In *20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS'01*. ACM, 2001.
- 50 Mihai Pătraşcu and Ryan Williams. On the possibility of faster SAT algorithms. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, pages 1065–1075. SIAM, 2010.
- 51 Franco P. Preparata and Michael Ian Shamos. *Computational Geometry — An Introduction*. Texts and Monographs in Computer Science. Springer, 1985.
- 52 Anastasios Sidiropoulos, Kritika Singhal, and Vijay Sridhar. Fractal Dimension and Lower Bounds for Geometric Problems. In *34th International Symposium on Computational Geometry, SoCG'18*, volume 99 of *LIPICs*, pages 70:1–70:14, 2018.
- 53 Warren D. Smith and Nicholas C. Wormald. Geometric Separator Theorem & Applications. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 232–243. IEEE, 1998.
- 54 Erik Jan van Leeuwen. *Optimization and Approximation on Systems of Geometric Objects*. PhD thesis, University of Amsterdam, 2009.
- 55 Kasturi R. Varadarajan. Weighted geometric set cover via quasi-uniform sampling. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 641–648. ACM, 2010.
- 56 Sergei Vassilvitskii and Mihalis Yannakakis. Efficiently computing succinct trade-off curves. *Theoretical Computer Science*, 348(2-3):334–356, 2005.
- 57 Daniel Vaz, Luís Paquete, and Aníbal Ponte. A note on the ε -indicator subset selection. *Theoretical Computer Science*, 499:113–116, 2013.

Simpler and Better Algorithms for Minimum-Norm Load Balancing

Deeparnab Chakrabarty

Dept. of Computer Science, Dartmouth, 9 Maynard St, Hanover, NH 03755, USA
deeparnab@dartmouth.edu

Chaitanya Swamy

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

Abstract

Recently, Chakrabarty and Swamy (STOC 2019) introduced the *minimum-norm load-balancing* problem on unrelated machines, wherein we are given a set J of jobs that need to be scheduled on a set of m unrelated machines, and a monotone, symmetric norm; We seek an assignment $\sigma : J \rightarrow [m]$ that minimizes the norm of the resulting load vector $\vec{\text{load}}_\sigma \in \mathbb{R}_+^m$, where $\vec{\text{load}}_\sigma(i)$ is the load on machine i under the assignment σ . Besides capturing all ℓ_p norms, symmetric norms also capture other norms of interest including top- ℓ norms, and ordered norms. Chakrabarty and Swamy (STOC 2019) give a $(38 + \varepsilon)$ -approximation algorithm for this problem via a general framework they develop for minimum-norm optimization that proceeds by first carefully reducing this problem (in a series of steps) to a problem called min-max ordered load balancing, and then devising a so-called deterministic oblivious LP-rounding algorithm for ordered load balancing.

We give a direct, and simple $4 + \varepsilon$ -approximation algorithm for the minimum-norm load balancing based on rounding a (near-optimal) solution to a novel convex-programming relaxation for the problem. Whereas the natural convex program encoding minimum-norm load balancing problem has a large non-constant integrality gap, we show that this issue can be remedied by including a key constraint that bounds the “norm of the job-cost vector.” Our techniques also yield a (essentially) 4-approximation for: (a) *multi-norm load balancing*, wherein we are given multiple monotone symmetric norms, and we seek an assignment respecting a given budget for each norm; (b) the best *simultaneous approximation factor* achievable for all symmetric norms for a given instance.

2012 ACM Subject Classification Theory of computation \rightarrow Approximation algorithms analysis; Theory of computation \rightarrow Convex optimization

Keywords and phrases Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.27

Funding *Deeparnab Chakrabarty*: Supported in part by NSF grant CCF-1813165.

Chaitanya Swamy: Supported in part by NSERC grant 327620-09 and an NSERC Discovery Accelerator Supplement Award.

1 Introduction

In the *minimum-norm load-balancing* (MinNormLB) problem, we are given a set J of n jobs, a set of m machines, and processing times $p_{ij} \geq 0$ for all $i \in [m]$ and $j \in J$. We use $[m]$ to denote $\{1, \dots, m\}$. We are also given a monotone, symmetric norm $f : \mathbb{R}^m \rightarrow \mathbb{R}_+$. Recall that by definition of norm, this means that f satisfies: (i) $f(x) = 0$ iff $x = 0$; (ii) $f(x + y) \leq f(x) + f(y)$ for all $x, y \in \mathbb{R}^m$ (triangle inequality); and (iii) $f(\lambda x) = |\lambda|f(x)$ for all $x \in \mathbb{R}^m, \lambda \in \mathbb{R}$ (homogeneity). (Properties (ii) and (iii) imply that f is convex.) Monotonicity means that $f(x) \leq f(y)$ for all $x, y \in \mathbb{R}^m$ such that $x_i(y_i - x_i) \geq 0$ for all $i \in [m]$; symmetry means that permuting the coordinates of x does not affect its norm, i.e., $f(x) = f(\{x_{\pi(i)}\}_{i \in [m]})$ for every $x \in \mathbb{R}^m$ and every permutation $\pi : [m] \mapsto [m]$.



© Deeparnab Chakrabarty and Chaitanya Swamy;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 27; pp. 27:1–27:12



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The goal is to find an assignment $\sigma : J \rightarrow [m]$ that minimizes the norm (under f) of the induced load vector. More precisely, an assignment σ induces the m -dimensional load vector $\vec{\text{load}}_\sigma \in \mathbb{R}_+^m$ where $\vec{\text{load}}_\sigma(i) := \sum_{j:\sigma(j)=i} p_{ij}$. The objective is to find σ that minimizes $f(\vec{\text{load}}_\sigma)$.

Besides ℓ_p -norms, monotone symmetric norms capture **Top- ℓ** norms – sum of ℓ largest coordinates in absolute value – and ordered norms (which are a nonnegative linear combination of **Top- ℓ** norms). The minimum-norm load-balancing problem was recently introduced by Chakrabarty and Swamy [8]. They develop a general framework for minimum-norm optimization problems based on reducing the problem to a special case called min-max ordered optimization, and devise a so-called deterministic oblivious rounding [8] to tackle the latter problem, which results in a $(38 + \varepsilon)$ -approximation algorithm for **MinNormLB**.

Our main result is a simpler $4(1 + \varepsilon)$ -approximation algorithm for **MinNormLB** that runs in time $\text{poly}(\text{input size}, \log(\frac{1}{\varepsilon}))$.

► **Theorem 1.** *One can achieve a $4(1 + \varepsilon)$ -approximation for **MinNormLB** in time $\text{poly}(\text{input size}, \log(\frac{1}{\varepsilon}))$, assuming we have a value-oracle and subgradient-oracle for the norm f . More generally, if we have ω -approximate value- and subgradient- oracles for f (see Section 4), then one can compute a $4(1 + 5\omega)(1 + \varepsilon)$ -approximation to **MinNormLB** in time $\text{poly}(\text{input size}, \log(\frac{1}{\varepsilon}))$.*

This is a substantial improvement over the approximation factor of 38 obtained in [8]. Moreover, our algorithm is also simpler and more direct than the one in [8]. Notably, our approximation factor is close to the best-known approximation factor (of 2) known for the ℓ_∞ norm (wherein **MinNormLB** becomes the classical minimum-makespan problem). Our algorithm proceeds by rounding the solution to a novel convex-programming relaxation of the problem. The convex program can be solved (approximately) using an (approximate) first-order oracle for f that returns the function value, and its subgradient at a given point.

Our techniques also yield a $4(1 + \varepsilon)$ -approximation for (see Section 5): (a) *multi-norm load balancing*, wherein we are given multiple monotone, symmetric norms and budgets for each norm, and we seek an assignment (approximately) respecting these budgets; and (b) the best *simultaneous approximation factor* achievable for all symmetric norms for a given instance.

Motivation and perspective

One of the reasons for studying **MinNormLB** is that it generalizes various load-balancing problems considered in the literature, and its study therefore yields a unified methodology for dealing with monotone, symmetric norms.

Load balancing under the ℓ_∞ norm, that is, minimizing the maximum load (also called the makespan) is a classical scheduling problem that has been extensively studied [18, 23, 10, 24, 6, 15] over the past three decades, both in its full generality for unrelated machines and for various special cases. The best known approximation factor for the unrelated-machines setting is still 2 [18], and it is *NP*-hard to obtain an approximation factor better than $3/2$ [18]. For general ℓ_p -norms, Azar and Epstein [3] obtain a 2-approximation, and improved guarantees have been obtained for constant p [3, 16, 19]. More recently, the load-balancing problem has also been considered for other monotone, symmetric norms. **Top- ℓ** - and ordered-norms have been proposed in the location-theory literature (see “Other related work”) as a means of interpolating between the ℓ_1 and ℓ_∞ norms (and an alternative to using ℓ_p norms), and motivated by this, Chakrabarty and Swamy [8] studied the **Top- ℓ** load-balancing

problem – minimize the total load on the ℓ most loaded machines – and the ordered load-balancing problem. They give a $(2 + \varepsilon)$ -approximation algorithm in both settings, and also (as noted earlier) devise a $(38 + \varepsilon)$ -approximation algorithm for an arbitrary monotone, symmetric norm.

For load balancing, there has been considerable interest in *simultaneous optimization*. Given an instance, the objective is to find an assignment that *simultaneously* approximates a large suite of objective functions. Building upon previous works [2, 4], Goel and Meyerson [11] describe a 2-approximation for the problem of simultaneously approximating all monotone symmetric norms in the *restricted assignment* setting. However, it is known that such an $O(1)$ -factor is *impossible* in the unrelated-machines setting [4, 11]. As a byproduct of their MinNormLB algorithm, in the unrelated-machines setting, Chakrabarty and Swamy [8] give an *instance-wise* $(38 + \varepsilon)$ -approximation to the best simultaneous approximation-factor possible for the instance. To elaborate, let $\alpha_{\mathcal{I}}^*$ denote the smallest factor for instance \mathcal{I} such that there exists a schedule that achieves an $\alpha_{\mathcal{I}}^*$ -approximation for all monotone, symmetric norms; [8] returns a schedule for \mathcal{I} that achieves a $38(1 + \varepsilon)\alpha_{\mathcal{I}}^*$ -approximation for all monotone, symmetric norms. As mentioned above, we devise an algorithm that for every instance \mathcal{I} returns a schedule that simultaneously achieves a $(4 + O(\varepsilon))\alpha_{\mathcal{I}}^*$ -approximation for all monotone, symmetric norms (see Theorem 13).

Our techniques

Since a norm is a convex function, a natural convex-programming relaxation for MinNormLB is to minimize the norm of the fractional load vector $\vec{L} = L(\vec{x}) := \{\sum_j p_{ij}x_{ij}\}_{i \in [m]}$, where the x_{ij} s are the usual variables denoting if job j is assigned to machine i , and we have the usual job-assignment constraints encoding that every job is assigned to some machine. This convex program, however, has a large integrality gap, even when f is the ℓ_∞ -norm due to the issue that the convex program could split a large job across multiple machines.

In the case of the ℓ_∞ norm (the makespan minimization problem), the typical way of circumventing the above issue is to “guess” the optimal value, say T , and add constraints to encode that no single job contributes more than T to the objective. The usual way of capturing this is to explicitly set $x_{ij} = 0$ if $p_{ij} > T$. A less common, and weaker, way of encoding this is to enforce that $\sum_i p_{ij}x_{ij} \leq T$ for all j , that is, the total processing time contribution of any job j across the machines cannot exceed T .

For an arbitrary (monotone, symmetric) norm, it is unclear how to extend either of the above approaches, since the contribution of a job to the objective is now a somewhat vague notion. One way to generalize things would be to encode (either explicitly or in the alternate weaker sense above) that the “norm” of the job-cost vector is at most T , where the job-cost vector is indexed by jobs and the cost for job j (under x) is $P_j := \sum_i p_{ij}x_{ij}$. But the norm f is defined over \mathbb{R}^m , whereas the job-cost vector lies in \mathbb{R}^n . For certain specific (families of) norms – e.g., ℓ_p -norms, top- ℓ norm, ordered norm – there is a natural version of the norm over \mathbb{R}^n ,¹ but what does such a constraint mean in general, and how can one encode this?

Our key insight, which leads to our convex program, is that one can capture the above consideration by examining the vector $\vec{P} \in \mathbb{R}^m$ comprising the *costs of the m most-costly jobs* and enforcing the constraint $f(\vec{P}) \leq T$; since f is monotone, this can be equivalently encoded as $f(\{P_j\}_{j \in S}) \leq T$ for all $S \subseteq J$ with $|S| = m$. It is not apparent that such a

¹ For ℓ_p -norms, a variant of this that considers the ℓ_p^p expression does work, but this crucially exploits the separability of ℓ_p^p [3].

constraint is valid, but we derive some insights about symmetric norms and show that this is indeed the case (see Theorem 3). This yields our convex program (CP), which can be solved efficiently (within ε additive error, for any $\varepsilon > 0$, in time $\text{poly}(\text{input size}, \ln(1/\varepsilon))$) using the ellipsoid method provided we have a value oracle and subgradient oracle for f .

Rounding a solution x to the convex program is now quite easy. Let $\vec{L} \in \mathbb{R}^m$ denote the load-vector arising from x . We use a filtering step to ensure that each job j is only assigned to machines i for which $p_{ij} \leq 2P_j$. This causes a factor-2 blowup in the machine loads. Now we use the rounding algorithm of Shmoys and Tardos [23] for the generalized assignment problem (GAP). The resulting assignment σ has load-vector at most $2\vec{L} + \vec{Z}$, where $\vec{Z} \in \mathbb{R}^m$ and $Z_i = \max_{j:\sigma(j)=i} p_{ij}$; the filtering step and our constraints ensure that $f(\vec{Z}) \leq 2T$, so $f(2\vec{L} + \vec{Z}) \leq 4T$. Our algorithm is much more direct than the one in [8]: it avoids the sequence of steps (and the associated approximation-factor losses) used in [8], wherein MinNormLB is reduced to a special case, called min-max ordered load balancing, which is then tackled by a deterministic oblivious rounding procedure.

Other related work

The algorithmic problem of finding minimum-norm solutions has also been investigated in the context of k -clustering, wherein the goal is to open k “facilities” in a metric space to serve a set of clients, and the cost vector induced by a solution is the vector of distances of clients to their nearest open facility. The setting of ℓ_p -norms, especially when $p \in \{1, 2, \infty\}$ (where the problem is called the k -{median, means, center} problem) has been extensively studied, and $O(1)$ -approximations are known in these settings [13, 9, 14, 1]. Top- ℓ and ordered norms have been proposed in the context of k -clustering in the Operations Research literature (see, e.g., [21, 17]), but constant-factor approximations for these norms were obtained quite recently [5, 7, 8]. Furthermore, Chakrabarty and Swamy [8] utilize their general framework to obtain an $O(1)$ -approximation for the k -clustering problem under any monotone, symmetric norm. We do not know of any alternate approach that works in the k -clustering setting.

2 A convex-programming relaxation

By possibly adding dummy jobs with zero processing times, we may assume without loss of generality that $n \geq m$. A natural convex program for MinNormLB has non-negative variables x_{ij} denoting if job j is assigned to machine i (or the extent of j assigned to i) with the constraint (1) encoding that every job is assigned to a machine. These x -variables define a load vector $\vec{L} = (L_i = L_i(x))_{i \in [m]}$ where $L_i(x) = \sum_{j \in J} p_{ij} x_{ij}$. The objective seeks to minimize $T := f(\vec{L})$. As noted earlier, this convex program has a large integrality gap (even when f is the ℓ_∞ norm). We *strengthen* the convex program as follows.

Given the x -assignment, define $P_j = P_j(x) := \sum_i p_{ij} x_{ij}$, which is the load incurred by the fractional solution for scheduling job j . Fix any subset $S \subseteq J$ with $|S| = m$. Note that this is well-defined since we have assumed $n \geq m$. This defines the m -dimensional vector $\vec{P}_S := \{P_j\}_{j \in S}$. We add the constraints (6) enforcing that $f(\vec{P}_S) \leq T$ for each such subset S . Throughout, we use i to index the machines in $[m]$, and j to index the jobs in J .

$$\begin{aligned}
 \min \quad & T & & \text{(CP)} \\
 \text{s.t.} \quad & \sum_i x_{ij} \geq 1 & \forall j \in J & \text{(1)} \\
 & x \geq 0 & & \text{(2)} \\
 & L_i = \sum_{j \in J} p_{ij} x_{ij} \quad \forall i \in [m] & & \text{(3)} \\
 & P_j = \sum_{i \in [m]} p_{ij} x_{ij} \quad \forall j \in J & & \text{(4)} \\
 & f(\vec{L}) \leq T & & \text{(5)} \\
 & f(\vec{P}_S) \leq T \quad \forall S \subseteq J, |S| = m & & \text{(6)}
 \end{aligned}$$

Let $\text{OPT} := \text{OPT}_{\text{CP}}$ denote the optimal value of (CP), and let O^* be the optimal value of the minimum-norm load-balancing problem. Since the x_{ij} -variables completely determine a solution to (CP), we will sometimes abuse notation and say that x is a feasible solution to (CP). We argue that (CP) is a valid relaxation. The proof uses the following simple observation about symmetric convex functions.

▷ **Claim 2.** Let $h : \mathbb{R}^m \rightarrow \mathbb{R}$ be a symmetric convex function. Let $v \in \mathbb{R}_+^m$, and $i, j \in [m]$. Let $w \in \mathbb{R}_+^m$ be the vector where $w_i = v_i + v_j$, $w_j = 0$, and $w_k = v_k$ otherwise. Then, $h(v) \leq h(w)$.

Proof. Consider the vector w' constructed in a symmetric fashion to w : set $w'_j = v_i + v_j$, $w'_i = 0$, and $w'_k = v_k$ otherwise. Observe that v is a convex combination of w and w' (we have $v = \frac{v_i}{v_i+v_j} \cdot w + \frac{v_j}{v_i+v_j} \cdot w'$), and $h(w) = h(w')$ since h is symmetric. By convexity and symmetry, $h(v) \leq \max\{h(w), h(w')\} = h(w)$. ◁

► **Theorem 3.** Constraints (6) are valid, and so for any instance of MinNormLB, we have $\text{OPT} \leq O^*$.

Proof. Let $\sigma^* : J \rightarrow [m]$ be an optimal assignment, so $f(\text{load}_{\sigma^*}) = O^*$. We now describe a feasible solution to (CP) with $T = O^*$. Set $x_{ij} = 1$ if $\sigma^*(j) = i$, and 0 otherwise. Clearly, constraints (1) hold. Note, $L_i = \text{load}_{\sigma^*}(i)$ for all i , and $P_j = p_{\sigma^*(j)j}$ for all j . Therefore, (5) holds with equality.

The interesting bit is to show that (6) holds. To that end, fix a subset $S \subseteq J$ of m jobs. Consider the load vector induced by jobs in S . That is, define $L'_i := \sum_{j \in S: \sigma^*(j)=i} p_{ij}$. Note that \vec{L} coordinate wise dominates \vec{L}' , so by monotonicity of f , we have $f(\vec{L}') \leq f(\vec{L}) = T$.

We argue that $f(\vec{P}_S) \leq f(\vec{L}')$, which will complete the proof. To see this, first note that if σ^* assigns the jobs in S to distinct machines, then \vec{P}_S is simply a permutation of \vec{L}' , so $f(\vec{P}_S) = f(\vec{L}')$. Otherwise, observe that \vec{L}' can be obtained from \vec{P}_S by applying the operation in Claim 2 to pairs of jobs in S assigned to the same machine; therefore, we have $f(\vec{P}_S) \leq f(\vec{L}')$. ◀

The proof above relied only on convexity, monotonicity, and symmetry of the function f . In Section 3 (see Theorem 4) we describe a rounding procedure which takes a feasible solution for (CP) and returns an assignment with a factor-4 blow-up in the objective. This will utilize the homogeneity of the norm f . In Section 4, we show how to (approximately) solve (CP) given an (approximate) first-order oracle for the underlying norm (see Theorem 9). Combining these two results yields Theorem 1.

3 The rounding algorithm

We now describe and analyze our simple rounding algorithm, which yields the following guarantee.

► **Theorem 4.** Given a feasible fractional solution $(x = \{x_{ij}\}_{i,j}, \vec{L}, \vec{P}, T)$ to (CP), there is a polynomial time algorithm to obtain a schedule σ with $f(\text{load}_{\sigma}) \leq 4T$.

Proof. First, we filter x . For every i, j , we set $\hat{x}_{ij} = 2x_{ij}$ if $p_{ij} \leq 2P_j$, and 0 otherwise. A standard Markov-inequality style argument shows that \hat{x} satisfies (1). Now we apply the Shmoys-Tardos GAP-rounding algorithm [23] to \hat{x} . This yields an assignment $\sigma : J \rightarrow [m]$ such that: for every job j , we have $\sigma(j) = i$ only if $\hat{x}_{ij} > 0$, and for every machine i , we have $\text{load}_{\sigma}(i) \leq \sum_{j \in J} p_{ij} \hat{x}_{ij} + Z_i \leq 2L_i + Z_i$, where $Z_i = \max_{j: \sigma(j)=i} p_{ij}$. Thus, $\text{load}_{\sigma} \leq 2\vec{L} + \vec{Z}$.

Let j_i be a maximum-length job assigned to machine i in σ , i.e., $\sigma(j_i) = i$ and $Z_i = p_{j_i}$. By our filtering step, we know that $Z_i \leq 2P_{j_i}$. Let $S = \{j_i : i \in [m]\}$. Then $\vec{Z} := (Z_i)_{i \in [m]} \leq 2\vec{P}_S$. By monotonicity, the triangle inequality, and homogeneity of f , we then obtain that

$$f(\text{load}_\sigma) \leq 2f(\vec{L}) + f(\vec{Z}) \leq 2T + 2f(\vec{P}_S) \leq 4T. \quad \blacktriangleleft$$

Interestingly, and notably, observe that the rounding procedure above is *oblivious to the norm* f : given a fractional solution x , the same rounding procedure works for all monotone, symmetric norms. This will be useful in Section 5, where we seek an assignment that is simultaneously good for multiple norms.

4 Solving the convex program

We now discuss how to solve the convex program (CP). To maintain the flow of reading, proofs of certain technical claims are deferred to Section 6. It is well known [20, 12] that we can efficiently solve a convex program $\min_{x \in S} h(x)$ (where $S \subseteq \mathbb{R}^n$ is convex) to within any additive error $\varepsilon > 0$ using the ellipsoid method provided that (we state things more precisely below): (i) S has non-zero volume and is contained in some ball; (ii) we have a separation oracle for S ; (iii) we have a *first-order* oracle for h that given input $x \in S$, returns $h(x)$, and a subgradient of h at x . More generally, we show that by utilizing the machinery of Shmoys and Swamy [22], even an approximate value and subgradient oracle suffices (see Theorem 9). This is particularly relevant since the norm and/or components of the subgradient vector may involve irrational numbers.

By scaling we may assume that all p_{ij} s are integers. Let O^* denote the optimal value for the MinNormLB instance. We can easily detect if $O^* = 0$, since this implies an assignment with 0 load on every machine. Therefore, we assume $O^* \geq 1$. It will be convenient to reformulate (CP) as follows. Let $\mathcal{P} := \{x \in \mathbb{R}^{[m] \times J} : \sum_i x_{ij} \geq 1 \ \forall j \in J, \ 0 \leq x_{ij} \leq 1 \ \forall i \in [m], j \in J\}$ denote the feasible region for the assignment variables.

$$\min g(x) := \max \left\{ f(L(\vec{x})), \max_{S \subseteq J: |S|=m} f(P(\vec{x})_S) \right\} \quad \text{s.t.} \quad x \in \mathcal{P}. \quad (\text{CP}')$$

Note that the x_{ij} s are the only variables above. Recall that OPT is the optimal value of (CP) (and (CP')).

We recall a few standard concepts from optimization. Let $h : \mathbb{R}^k \mapsto \mathbb{R}$ and let $\|u\|$ denote the ℓ_2 norm of u .

- We say that h has *Lipschitz constant* (at most) K if $|h(v) - h(u)| \leq K\|v - u\|$ for all $u, v \in \mathbb{R}^k$.
- We say that $d \in \mathbb{R}^k$ is a *subgradient* of h at $u \in \mathbb{R}^k$ if we have $h(v) - h(u) \geq d \cdot (v - u)$ for all $v \in \mathbb{R}^k$. We say that \hat{d} is an ω -*subgradient* of h at $u \in \mathbb{R}^k$ if for every $v \in \mathbb{R}^k$, we have $h(v) - h(u) \geq \hat{d} \cdot (v - u) - \omega h(u)$; we call this the approximate-subgradient inequality.
- An ω -*first-order oracle* for h is an algorithm that at any point $u \in \mathbb{R}^k$, returns an estimate est such that $h(u) \leq \text{est} \leq (1 + \omega)h(u)$, and an ω -subgradient of h at u .

(In the optimization literature, the notions of approximate first-order oracle and approximate subgradient typically involve additive errors; since our problems are scale-invariant, multiplicative approximations, where the error at u is measured relative to $h(u)$, are more apt here.)

We remark that since f is a norm, an ω -subgradient \hat{d} of f at u also yields an estimate of $f(u)$ as follows: taking $v = \vec{0}$ and $v = 2u$ respectively in the approximate-subgradient inequality, we obtain the bounds $\hat{d} \cdot u \geq (1 - \omega)f(u)$ and $\hat{d} \cdot u \leq (1 + \omega)f(u)$. (Thus, an ω -first-order oracle for f boils down to an ω -subgradient oracle for f .)

By input size, we mean the total encoding length of the p_{ij} s. It is easy to separate over \mathcal{P} , and easy to find radii R , and $0 < V \leq 1$ such that $\mathcal{P} \subseteq B(0, R) := \{x : \|x\| \leq R\}$, \mathcal{P} contains a ball of radius V , and $\log(\frac{R}{V}) = \text{poly}(m, n)$. In particular, $R = \sqrt{mn}$ suffices, and \mathcal{P} contains a ball of radius $V = \frac{0.5}{m}$ around the point x with $x_{ij} = \frac{1.5}{m}$ for all i, j . (We may assume $m \geq 2$ as otherwise the problem is trivial.) Throughout, we use K_f to denote an efficiently-computable upper bound on the Lipschitz constant of f ; Lemma 8 shows how to obtain this. Given a bound on the Lipschitz constant of f , one can compute an upper bound on the Lipschitz constant of g .

▷ **Claim 5.** The Lipschitz constant of g is at most $K = \sqrt{mn} \cdot \max_{i,j} p_{ij} \cdot K_f$.

► **Theorem 6** (Follows from [20]; see also [12]). *Let alg be a first-order oracle for f . Then, for any $\eta > 0$, we can compute $x^* \in \mathcal{P}$ such that $g(x^*) \leq \text{OPT} + \eta$ in $\text{poly}(\text{input size}, \log(\frac{K_f R}{\eta V}))$ time and using $\text{poly}(\text{input size}, \log(\frac{K_f R}{\eta V}))$ calls to alg .*

Theorem 6 follows from the ellipsoid method for convex optimization, due to the bound on the Lipschitz constant of g obtained from Claim 5, and since one can use alg to obtain a first-order oracle for g . We next use [22] to obtain a stronger result that utilizes only an approximate first-order oracle for f .

► **Theorem 7** (Lemma 4.5 in [22] paraphrased). *Consider a convex optimization problem: $\min_{x \in \mathcal{P}} h(x)$. Let K_h be a known bound on the Lipschitz constant of h . Let $\omega < 1$ and $\eta > 0$. In $\text{poly}(m, n, \log(\frac{K_h R}{V \eta}))$ time and using $\text{poly}(m, n, \log(\frac{K_h R}{V \eta}))$ calls to an ω -first-order oracle for h , one can compute a solution $x^* \in \mathcal{P}$ such that $h(x^*) \leq \frac{1+\omega}{1-\omega} \cdot (\min_{x \in \mathcal{P}} h(x) + \eta)$.*

To utilize Theorem 7 to solve (CP), we show how to obtain an approximate first-order oracle for g given one for f . Also, in order to convert the additive error in Theorem 7 (and Theorem 6) into a multiplicative guarantee, we show how to obtain a lower bound lb on O^* such that K_f/lb is small.

► **Lemma 8.** *Let alg be an ω -first-order oracle for f (where $\omega < 1$).*

- *We can obtain a 2ω -first-order oracle for g using $O(1)$ calls to alg .*
- *Using alg , we can efficiently compute $\text{lb} \leq O^*$, and an upper bound K_f on the Lipschitz constant of f such that $\frac{K_f}{\text{lb}} \leq 2\sqrt{m}$.*

► **Theorem 9.** *Let alg be an ω -first-order oracle for f with $\omega \leq \frac{1}{10}$. Given a MinNormLB instance with optimum value O^* , there is an algorithm that, for any $\varepsilon > 0$, computes a feasible solution x^* to (CP) of objective value $g(x^*) \leq (1 + 5\omega)(1 + \varepsilon)O^*$. The algorithm runs in $\text{poly}(\text{input size}, \log(\frac{1}{\varepsilon}))$ time and makes $\text{poly}(\text{input size}, \log(\frac{1}{\varepsilon}))$ calls to alg .*

Proof. This follows by combining Theorem 7 and Lemma 8. Recall that we are assuming that $O^* \geq 1$. By part 8 of Lemma 8, we can compute a 2ω -first-order oracle for g . We use part 8 of Lemma 8 to obtain lb and K_f . Now we apply Theorem 7 to the problem $\min_{x \in \mathcal{P}} g(x)$, taking $\eta = \varepsilon \text{lb}$. The point x^* returned satisfies $g(x^*) \leq \frac{1+2\omega}{1-2\omega} \cdot (\text{OPT} + \varepsilon \text{lb}) \leq (1 + 5\omega)(1 + \varepsilon)O^*$.

Recall that $\log(R/V) = \text{poly}(m, n)$. Since we have an upper bound K on the Lipschitz constant of g , where $\log K = \text{poly}(\text{input size}) \cdot \log K_f$ (Claim 5), the running time and number of calls to the first-order oracle for g (and hence alg) is at most $\text{poly}(\text{input size}, \log(\frac{1}{\varepsilon}))$. ◀

5 Extensions: multi-norm load balancing and simultaneous approximation

5.1 Multi-norm load balancing

In the *multi-norm load-balancing problem*, we are given a load-balancing instance $(J, m, \{p_{ij}\}_{i \in [m], j \in J})$, multiple monotone, symmetric norms f_1, \dots, f_k , and budgets T_1, \dots, T_k for these norms respectively. The goal is to find an assignment $\sigma : J \rightarrow [m]$ such that $f_r(\vec{\text{load}}_\sigma) \leq T_r$ for all $r \in [k]$. Our approximation guarantee extends easily to this problem.

► **Theorem 10.** *Let $(J, m, \{p_{ij}\}_{i \in [m], j \in J})$ be a load-balancing instance. Let f_1, \dots, f_k be k monotone, symmetric norms, with associated budgets T_1, \dots, T_k . Given an ω -first-order oracle for each norm, for any $\varepsilon > 0$, in $\text{poly}(\text{input size}, k, \log(1/\varepsilon))$ time, one can either determine that there is no feasible solution to the multi-norm load-balancing problem, or return an assignment $\sigma : J \rightarrow [m]$ such that $f_r(\vec{\text{load}}_\sigma) \leq 4(1 + 7\omega)(1 + \varepsilon)T_r$ for all $r \in [k]$.*

The convex-programming relaxation for this problem is a variant of (CP) where there is no objective function, and constraints (5), (6) are replaced with

$$f_r(\vec{L}) \leq T_r, \quad f_r(\vec{P}_S) \leq T_r \quad \forall S \subseteq J : |S| = m, \quad \forall r = 1, \dots, k \quad (7)$$

Let (Multi-CP) denote the resulting feasibility problem: find (x, \vec{L}, \vec{P}) satisfying (1)–(4), and (7). As noted earlier, the rounding procedure in Section 3 is *oblivious* to the underlying norm, and so our task boils down to finding an (approximately) feasible solution to (Multi-CP).

In order to solve (Multi-CP), as with (CP), it will be convenient to move the nonlinear constraints to the objective and consider the following reformulation:

$$\min q(x) := \max \left\{ \max_{r \in [k]} \frac{f_r(\vec{L}(x))}{T_r}, \max_{r \in [k]} \max_{S \subseteq J : |S| = m} \frac{f_r(\vec{P}(x)_S)}{T_r} \right\} \quad \text{s.t.} \quad (1), (2). \quad (\text{MNCP})$$

Observe that finding a feasible solution to (Multi-CP) is equivalent to finding a feasible solution to (MNCP) with objective value at most 1. As before, we may assume that the p_{ij} s are integers, and can determine if there is an assignment σ such that $\vec{\text{load}}_\sigma = \vec{0}$ (which clearly satisfies (7)). So assume otherwise. We prove the following.

► **Theorem 11.** *Let alg_r be an ω -first-order oracle for f_r for all $r \in [k]$, where $\omega \leq \frac{1}{18}$. For any $\varepsilon > 0$, in $\text{poly}(\text{input size}, \log(\frac{1}{\varepsilon}))$ time and using $\text{poly}(\text{input size}, \log(\frac{1}{\varepsilon}))$ calls to each alg_r oracle, we can determine that either (Multi-CP) is infeasible, or compute $x^* \in \mathcal{P}$ such that $q(x^*) \leq (1 + 7\omega)(1 + \varepsilon)$.*

Using Theorem 11, for any $\varepsilon > 0$, we can determine in time $\text{poly}(\text{input size}, \log(\frac{1}{\varepsilon}))$ that (Multi-CP) is infeasible, or return a fractional assignment x^* satisfying

$$f_r(L(\vec{x}^*)) \leq \kappa T_r, \quad f_r(P(\vec{x}^*)_S) \leq \kappa T_r \quad \forall S \subseteq J : |S| = m, \quad \forall r = 1, \dots, k$$

where $\kappa = (1 + 7\omega)(1 + \varepsilon)$. As noted earlier, the rounding procedure in Section 3 is *oblivious* to the underlying norm, and so by utilizing this to round x^* , we obtain an assignment σ such that $f_r(\vec{\text{load}}_\sigma) \leq 4\kappa T_r$ for all $r \in [k]$. This yields Theorem 10.

In the rest of this section, we discuss the proof of Theorem 11. If the multi-norm problem is feasible, we must have $T_r \geq f_r(e_1)$ for all $r \in [k]$. We assume in the sequel that T_r is at least the estimate of $f_r(e_1)$ returned by alg_r scaled by $(1 + \omega)$, for all $r \in [k]$; if this

does not hold, then we declare infeasibility. Given this, the proof of Lemma 8.8 shows that $K_r = (1 + \omega)\sqrt{m} \cdot T_r$ is an upper bound on the Lipschitz constant of f_r , for all $r \in [k]$. We assume this bound in the sequel. Similar to Claim 5 and Lemma 8, we show that the Lipschitz constant of q can be bounded in terms of the K_r s, and we can obtain a 2ω -first-order oracle for q using the alg_r oracles.

► **Lemma 12.** (i) *The Lipschitz constant of q is bounded by $K = \text{poly}(m, n, \max_{i,j} p_{ij})$. (ii) We can obtain a 2ω -first order oracle for q by making $O(1)$ calls to alg_r for each $r \in [k]$.*

Proof of Theorem 11. We utilize Lemma 12 in conjunction with Theorem 7. Part (ii) of Lemma 12 shows how to obtain a 2ω -first-order oracle, alg , for q . So invoking Theorem 7 with $\eta = \varepsilon$, and the bound K on the Lipschitz constant of q obtained from part (i) of Lemma 12, we obtain $\bar{x} \in \mathcal{P}$ such that

$$q(\bar{x}) \leq \frac{1 + 2\omega}{1 - 2\omega} \left(\min_{x \in \mathcal{P}} q(x) + \eta \right). \quad (8)$$

The running time is $\text{poly}(\text{input size}, \log(\frac{1}{\varepsilon}))$ (since $\log(R/V)$, $\log K = \text{poly}(\text{input size})$), and this is also a bound on the number of calls to the alg_r oracles. Using alg , we obtain an estimate est such that $q(\bar{x}) \leq \text{est} \leq (1 + 2\omega)q(\bar{x})$. If $\text{est} > \frac{(1+2\omega)^2}{1-2\omega} \cdot (1 + \eta)$, then (8) implies that $(\min_{x \in \mathcal{P}} q(x)) > 1$, and so (Multi-CP) is infeasible. Otherwise, taking $x^* = \bar{x}$, we obtain that $q(x^*) \leq \text{est} \leq \frac{(1+2\omega)^2}{1-2\omega} \cdot (1 + \varepsilon) \leq (1 + 7\omega)(1 + \varepsilon)$ since $\omega \leq \frac{1}{18}$. ◀

5.2 Simultaneous approximation

Given a load-balancing instance $\mathcal{I} = (J, m, \{p_{ij}\}_{i \in [m], j \in J})$, let $\alpha_{\mathcal{I}}^*$ be the smallest α such that there *exists* an assignment σ^* satisfying $f(\text{load}_{\sigma^*}) \leq \alpha (\min_{\sigma: J \rightarrow [m]} f(\text{load}_{\sigma}))$ for every monotone, symmetric norm. That is, $\alpha_{\mathcal{I}}^*$ is the best *simultaneous approximation factor* achievable on instance \mathcal{I} . Instead of seeking absolute bounds on $\alpha_{\mathcal{I}}^*$ over a class of instances [2, 4, 11], as discussed in [8], another pertinent problem is to seek *instance-wise* guarantees: given an instance \mathcal{I} , we want to find a polytime-computable assignment $\bar{\sigma}$ such that, for some factor $\gamma \geq 1$, we have $f(\text{load}_{\bar{\sigma}}) \leq \gamma \alpha_{\mathcal{I}}^* (\min_{\sigma: J \rightarrow [m]} f(\text{load}_{\sigma}))$ for every monotone, symmetric norm; i.e., the simultaneous approximation factor of $\bar{\sigma}$ at most γ times the best simultaneous approximation factor achievable for \mathcal{I} .

Our techniques coupled with insights from [11, 8] yields a $4(1 + O(\varepsilon))$ -approximation to the best simultaneous approximation factor, in time $\text{poly}(\text{input size}, (\frac{m}{\varepsilon})^{O(1/\varepsilon)})$. To obtain this guarantee, following [11, 8], incurring a $(1 + \varepsilon)$ -factor loss, it suffices to obtain a 4-approximation to the best simultaneous-approximation achievable for **Top- ℓ -norms** – $\text{Top-}\ell(x) := \max_{S \subseteq [m]: |S|=\ell} \sum_{i \in S} |x_i|$ – for the $O(\log m)$ indices ℓ in $\text{POS} := \{\min\{\lceil (1 + \varepsilon)^s \rceil, m\} : s \geq 0\}$. If we knew the optimal value opt_{ℓ} for each such **Top- ℓ** norm, then we can set a budget $T_{\ell} = \alpha \text{opt}_{\ell}$ for each $\ell \in \text{POS}$, and utilize our result for multi-norm load balancing to do a binary search for α . Importantly, notice that the resulting feasibility problem (Multi-CP) can now be cast as an *linear-programming* feasibility problem, since a budget constraint of the form $\text{Top-}\ell(\vec{v}) \leq T_{\ell}$ can be modeled using exponentially many linear constraints that one can separate over. Thus, this would yield a $4(1 + \varepsilon)$ -approximation. To make this idea work, we enumerate all choices for the opt_{ℓ} values in powers of $(1 + \varepsilon)$. As argued in [8], there are at most $\text{poly}(\text{input size}, (\frac{m}{\varepsilon})^{O(1/\varepsilon)})$ candidates to enumerate over, and this yields the stated guarantee.

► **Theorem 13.** *Given a load-balancing instance $\mathcal{I} = (J, m, \{p_{ij}\}_{i \in [m], j \in J})$, let α_I^* be the smallest α such that there is an assignment σ^* satisfying $f(\vec{\text{load}}_{\sigma^*}) \leq \alpha (\min_{\sigma: J \rightarrow [m]} f(\vec{\text{load}}_{\sigma}))$ for every monotone, symmetric norm f . In $\text{poly}(\text{input size}, (\frac{m}{\varepsilon})^{O(1/\varepsilon)})$ time, we can find an assignment $\hat{\sigma}$ such that we have $f(\vec{\text{load}}_{\hat{\sigma}}) \leq (4 + O(\varepsilon)) \alpha_I^* (\min_{\sigma: J \rightarrow [m]} f(\vec{\text{load}}_{\sigma}))$ for every monotone, symmetric norm f .*

6 Proofs from Sections 4 and 5

Proof of Claim 5. The bound follows easily from the definition of g . Let $x, y \in \mathbb{R}^{[m] \times J}$. Let $\vec{L}, \vec{L}' \in \mathbb{R}^m$ be the load vectors induced by x, y respectively; let \vec{P}_S, \vec{P}'_S be the job-cost vectors for the jobs in S induced by x, y respectively. Then, $g(y) - g(x) \leq \max\{f(L') - f(L), \max_{S \subseteq J: |S|=m} f(P'_S) - f(P_S)\}$. So $g(y) - g(x) \leq K_f \|L' - L\|_2$ or $g(y) - g(x) \leq K_f \|P'_S - P_S\|$ for some $S \subseteq J$ with $|S| = m$. Let $p_{\max} := \max_{i,j} p_{ij}$. In the former case, we have $g(y) - g(x) \leq K_f p_{\max} \sum_{i,j} |y_{ij} - x_{ij}| \leq \sqrt{mn} \cdot K_f p_{\max} \|y - x\|_2$; the same bound also applies in the latter case. This shows that $K = \sqrt{mn} \cdot K_f p_{\max}$ is a bound on the Lipschitz constant of g . ◀

The following claim will be useful in proving part 8 of Lemma 8, as also part (ii) of Lemma 12.

▷ **Claim 14.** Let $h : \mathbb{R}^n \mapsto \mathbb{R}$ be defined by $h(x) := \max_{r \in [k]} h_r(x)$, where $h_r : \mathbb{R}^n \mapsto \mathbb{R}$ is convex for all $r \in [k]$. Let alg_r be an ω -first order oracle for h_r for all $r \in [k]$ (where $\omega < 1$).

- One can obtain a 2ω -first order oracle for h using $O(1)$ calls to $\text{alg}_1, \dots, \text{alg}_k$.
- More generally, suppose that given $x \in \mathbb{R}^n$, one can identify $I(x) \subseteq [k]$ such that $h(x) = \max_{r \in I(x)} h_r(x)$. Then, one can compute a 2ω -first-order oracle for h that, on input $x \in \mathbb{R}^n$, makes $O(1)$ calls to alg_r for all $r \in I(x)$.

Proof. We focus on proving part (i); part (ii) follows from a very similar argument. Fix $x \in \mathbb{R}^n$. For every $r \in [k]$, we call alg_r to obtain an estimate est^r of $h_r(x)$. We set the estimate for $h(x)$ to be $\text{est} := \max_{r \in [k]} \text{est}^r$. From the properties of est^r , it is easy to see that $h(x) \leq \text{est} \leq (1 + \omega)h(x)$.

Let d^r be the ω -subgradient of f_r at x returned by alg_r . Let $s \in [k]$ be such that $\text{est} = \text{est}^s$. We set $\mu = d^s$. We now argue that μ is a 2ω -subgradient of h at x . Consider any $y \in \mathbb{R}^n$. We have

$$\begin{aligned} \mu^T(y - x) &= (y - x)^T d^s \leq h_s(y) - h_s(x) + \omega h_s(x) \leq h(y) - \frac{1-\omega}{1+\omega} \cdot \text{est}^s = h(y) - \frac{1-\omega}{1+\omega} \cdot \text{est} \\ &\leq h(y) - \frac{1-\omega}{1+\omega} \cdot h(x) \leq h(y) - (1 - 2\omega)h(x). \end{aligned}$$

The first two inequalities follow due to the fact that (est^s, d^s) was returned by the ω -first order oracle for h_s ; the next equality follows from the definition of index s ; and the penultimate inequality follows since $\text{est} \geq h(x)$ as established earlier.

The proof of the more general statement in (ii) is essentially identical: on input x , we now run alg^r for all $r \in I(x)$; we set $\text{est} = \max_{r \in I(x)} \text{est}^r$, and $d = d^s$, where $s \in I(x)$ is an index such that $\text{est} = \text{est}^s$. ◀

Proof of Lemma 8. For part 8, fix $x \in \mathbb{R}^{[m] \times J}$. Recall that $P_j = P_j(x) := \sum_i p_{ij} x_{ij}$. Let S^* be the set of m jobs with the highest P_j values. Let $\vec{L} = L(x)$ and $\vec{P}_{S^*} = P(x)_{S^*}$. Then, $g(x) = \max\{f(\vec{L}), f(\vec{P}_{S^*})\}$. Observe that alg can be used to obtain an ω -first-order oracle for both $f(L(x))$ and $f(P(x)_{S^*})$. Thus, by using Claim 14 (ii), we obtain a 2ω -first-order oracle for g using $O(1)$ calls to alg .

We now justify the observation. A $(1 + \omega)$ -approximate value oracle is obtained by simply calling `alg` to obtain estimates of $f(\vec{L})$ and $f(\vec{P}_{S^*})$. Let $d^L = (d_i^L)_{i \in [m]}$, and $d^P = (d_j^P)_{j \in S^*}$ be the ω -subgradients of f at \vec{L} at \vec{P}_{S^*} respectively returned by `alg`.

$$\text{For all } i \in [m], j \in J, \text{ define } \quad \beta_{ij} = p_{ij}d_i^L, \quad \gamma_{ij} = \begin{cases} p_{ij}d_j^P & \text{if } j \in S^*; \\ 0 & \text{otherwise.} \end{cases}$$

Then, for any $y \in \mathbb{R}^{[m] \times J}$, we have $\beta^T(y - x) = \sum_{i,j} d_i^L p_{ij}(y_{ij} - x_{ij}) = (L(\vec{y}) - L(\vec{x}))^T d^L$ showing that β is an ω -subgradient of $f(L(\vec{\cdot}))$ at x . Similarly, $\gamma^T(y - x) = (P(\vec{y})_{S^*} - P(\vec{x})_{S^*})^T d^P$ showing that γ is an ω -subgradient of $f(P(\vec{\cdot}))$ at x .

For part 8, Let σ^* be an optimal assignment. Since we are assuming that $O^* \geq 1$, we have $\text{load}_{\sigma^*}(i) \geq 1$ for some $i \in [m]$. Let $e_i \in \mathbb{R}^m$ be the vector with 1 in coordinate i and 0s everywhere else. Then, $O^* \geq f(e_i)$. Let `lb` be the estimate of $f(e_i)$ obtained by `alg` scaled down by $(1 + \omega)$. So we have $f(e_i)/(1 + \omega) \leq \text{lb} \leq O^*$. Consider any $x, y \in \mathbb{R}^m$. We have $y = x + \sum_{i=1}^m (y_i - x_i)e_i$, so by the triangle inequality and symmetry, we have $|f(y) - f(x)| \leq \sum_{i=1}^m |y_i - x_i|f(e_i)$. Therefore, $|f(y) - f(x)| \leq (1 + \omega)\text{lb} \sum_{i=1}^m |y_i - x_i| \leq (1 + \omega)\sqrt{m} \cdot \text{lb} \cdot \|y - x\|$. So we can set $K_f = (1 + \omega)\sqrt{m} \cdot \text{lb}$. ◀

Proof of Lemma 12. Part (i) follows by applying Claim 5 to each norm f_r , and since the Lipschitz constant of the maximum of a collection of functions is bounded by the maximum of the Lipschitz constants of the functions in the collection. Let $p_{\max} = \max_{i,j} p_{ij}$. By Claim 5, for each $r \in [k]$, and $S \subseteq J$ with $|S| = m$, both $f_r(L(\vec{x}))/T_r$ and $f_r(P(\vec{x})_S)/T_r$ have Lipschitz constant at most $\sqrt{mn} \cdot p_{\max} \cdot K_r/T_r \leq (1 + \omega)m\sqrt{n}p_{\max}$. Hence, the Lipschitz constant of q is at most $K = (1 + \omega)m\sqrt{n}p_{\max}$.

For part (ii), we mimic the proof of part 8 of Lemma 8. Fix $x \in \mathbb{R}^{[m] \times J}$. Let S^* be the set of m jobs with the highest $P_j(x)$ values, where $P_j(x) := \sum_i p_{ij}x_{ij}$. Let $\vec{L} = L(\vec{x})$ and $\vec{P}_{S^*} = P(\vec{x})_{S^*}$. Then,

$$q(x) = \max \left\{ \max_{r \in [k]} f_r(\vec{L})/T_r, \max_{r \in [k]} f_r(\vec{P}_{S^*})/T_r \right\}.$$

As in the proof of Lemma 8 8, for each $r \in [k]$, we can use `algr` to obtain an ω -first-order oracle for $f_r(L(\vec{x}))/T_r$ and $f_r(P(\vec{x})_{S^*})/T_r$. Thus, by using Claim 14 (ii), we obtain a 2ω -first-order oracle for q using $O(1)$ calls to `algr`, for each $r \in [k]$. ◀

References

- 1 Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k -means and Euclidean k -median by primal-dual algorithms. In *Proceedings, FOCS*, pages 61–72, 2017.
- 2 Noga Alon, Yossi Azar, Gerhard Woeginger, and Tal Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.
- 3 Yossi Azar and Amir Epstein. Convex programming for scheduling unrelated parallel machines. In *Proceedings, STOC*, pages 331–337, 2005.
- 4 Yossi Azar, Leah Epstein, Yossi Richter, and Gerhard J. Woeginger. All-norm approximation algorithms. *J. Algorithms*, 52(2):120–133, 2004.
- 5 Jarosław Byrka, Krzysztof Sornat, and Joachim Spoerhase. Constant-factor approximation for ordered k -median. In *Proceedings, STOC*, pages 620–631, 2018.
- 6 Deeparnab Chakrabarty, Sanjeev Khanna, and Shi Li. On $(1, \varepsilon)$ -restricted assignment makespan minimization. In *Proceedings, SODA*, pages 1087–1101, 2015.

- 7 Deeparnab Chakrabarty and Chaitanya Swamy. Interpolating between k -median and k -center: Approximation Algorithms for Ordered k -median. In *Proceedings, ICALP*, pages 29:1–29:14, 2018.
- 8 Deeparnab Chakrabarty and Chaitanya Swamy. Approximation algorithms for minimum norm and ordered optimization problems. In *Proceedings, STOC*, pages 126–137, 2019.
- 9 Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k -median problem. *J. Comput. System Sci.*, 65(1):129–149, 2002.
- 10 Tomáš Ebenlendr, Marek Krčál, and Jiří Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. *Algorithmica*, 68(1):62–80, 2014.
- 11 Ashish Goel and Adam Meyerson. Simultaneous optimization via approximate majorization for concave profits or convex costs. *Algorithmica*, 44(4):301–323, 2006.
- 12 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1988.
- 13 Dorit S. Hochbaum and David B. Shmoys. A Best Possible Heuristic for the k -Center Problem. *Math. Oper. Res.*, 10(2):180–184, 1985.
- 14 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 (JACM)*, 48(2):274–296, 2001.
- 15 Klaus Jansen and Lars Rohwedder. On the configuration-LP of the restricted assignment problem. In *Proceedings, SODA*, pages 2670–2678, 2017.
- 16 V. S. Kumar, Madhav V Marathe, Srinivasan Parthasarathy, and Aravind Srinivasan. A unified approach to scheduling on unrelated parallel machines. *Journal of the ACM (JACM)*, 56(5):28, 2009.
- 17 G. Laporte, S. Nickel, and F. S. da Gama. *Location Science*. Springer, 2015.
- 18 Jan K. Lenstra, David B. Shmoys, and Éva. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Programming*, 46(1-3):259–271, 1990.
- 19 Konstantin Makarychev and Maxim Sviridenko. Solving optimization problems with diseconomies of scale via decoupling. In *Proceedings, FOCS*, pages 571–580, 2014.
- 20 A. Nemirovski and Yudin D. *Problem complexity and method efficiency in optimization*. John Wiley and Sons, 1983.
- 21 S. Nickel and J. Puerto. *Location Theory: A Unified Approach*. Springer Science & Business Media, 2005.
- 22 David B. Shmoys and Chaitanya Swamy. An Approximation Scheme for Stochastic Linear Programming and Its Application to Stochastic Integer Programs. *Journal of the ACM*, 53(6):978–1012, 2006.
- 23 David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical programming*, 62(1-3):461–474, 1993.
- 24 Ola Svensson. Santa Claus schedules jobs on unrelated machines. *SIAM Journal on Computing*, 41(5):1318–1341, 2012.

On Computing Centroids According to the p -Norms of Hamming Distance Vectors

Jiehua Chen

Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Warsaw, Poland
jiehua.chen2@gmail.com

Danny Hermelin

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

Manuel Sorge

Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Warsaw, Poland
manuel.sorge@mimuw.edu.pl

Abstract

In this paper we consider the p -NORM HAMMING CENTROID problem which asks to determine whether some given strings have a centroid with a bound on the p -norm of its Hamming distances to the strings. Specifically, given a set S of strings and a real k , we consider the problem of determining whether there exists a string s^* with $(\sum_{s \in S} d^p(s^*, s))^{1/p} \leq k$, where $d(\cdot)$ denotes the Hamming distance metric. This problem has important applications in data clustering and multi-winner committee elections, and is a generalization of the well-known polynomial-time solvable CONSENSUS STRING ($p = 1$) problem, as well as the NP-hard CLOSEST STRING ($p = \infty$) problem.

Our main result shows that the problem is NP-hard for all fixed rational $p > 1$, closing the gap for all rational values of p between 1 and ∞ . Under standard complexity assumptions the reduction also implies that the problem has no $2^{o(n+m)}$ -time or $2^{o(k \frac{p}{p+1})}$ -time algorithm, where m denotes the number of input strings and n denotes the length of each string, for any fixed $p > 1$. The first bound matches a straightforward brute-force algorithm. The second bound is tight in the sense that for each fixed $\varepsilon > 0$, we provide a $2^{k \frac{p}{p+1} + \varepsilon}$ -time algorithm. In the last part of the paper, we complement our hardness result by presenting a fixed-parameter algorithm and a factor-2 approximation algorithm for the problem.

2012 ACM Subject Classification Theory of computation \rightarrow Computational complexity and cryptography; Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Approximation algorithms analysis; Theory of computation \rightarrow Facility location and clustering; Theory of computation \rightarrow Lower bounds and information complexity; Theory of computation \rightarrow Design and analysis of algorithms; Mathematics of computing \rightarrow Combinatorial optimization

Keywords and phrases Strings, Clustering, Multiwinner Election, Hamming Distance

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.28

Related Version A full version of the paper is available at <https://arxiv.org/abs/1807.06469>.

Funding Research started while both JC and MS were with Ben-Gurion University of the Negev, Beer-Sheva, Israel. This work is supported 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, the Israel Science Foundation (grant no. 551145/14), and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement numbers 677651 (JC) and 714704 (MS).



© Jiehua Chen, Danny Hermelin, and Manuel Sorge;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 28; pp. 28:1–28:16
Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The *Hamming distance between two strings* of equal length is the number of positions at which the corresponding symbols in the strings differ. In other words, it measures the number of substitutions of symbols required to change one string into the other, or the number of errors that could have transformed one string into the other. This is perhaps the most fundamental string metric known, named after Richard Hamming who introduced the concept in 1950 [24].

While Hamming distance has a variety of applications in a plethora of different domains, a common usage for it appears when clustering data of various sorts. Here, one typically wishes to cluster the data into groups that are centered around some centroid, where the notion of centroid varies from application to application. Two prominent examples in this context are: **Consensus String**, where the centroid has a bound on the sum of its (Hamming) distance to all strings, and

Closest String, where the centroid has a bound on the maximum distance to all strings.

In functional analysis terms, these two problems can be formalized using the p -norms of the Hamming distance vectors associated with the clusters. That is, if $S \subseteq \{0, 1\}^n$ is a cluster and $s^* \in \{0, 1\}^n$ is its centroid, then the p -norm of the corresponding Hamming distance vector is defined by

$$\|(s^*, S)\|_p := \left(\sum_{s \in S} d^p(s^*, s) \right)^{1/p},$$

where $d(s^*, s) = |\{i: s^*[i] \neq s[i], 1 \leq i \leq n\}|$ denotes the Hamming distance between s^* and s . Using this notation, we can formulate **CONSENSUS STRING** as the problem of finding a centroid s^* with a bound on $\|(s^*, S)\|_1$ for a given set S of strings, while **CLOSEST STRING** can be formulated as the problem of finding a centroid s^* with a bound on $\|(s^*, S)\|_\infty$.

The following cluster S with 5 strings, each of length 7, shows that for different p , we indeed obtain different optimal centroids. For each $p \in \{1, 2, \infty\}$, string s_p^* is an optimal p -norm centroid but it is not an optimal q -norm centroid, where $q \in \{1, 2, \infty\} \setminus \{p\}$. Moreover, one can verify that s_2^* is the only optimal 2-norm centroid and no optimal ∞ -norm centroid is an optimal 2-norm centroid.

$S :$		p		
	centroid	$\ \cdot\ _1$	$\ \cdot\ _2$	$\ \cdot\ _\infty$
1111 111	$s_1^* = 0000\ 000$	14	$\sqrt{68}$	7
1111 000	$s_2^* = 0011\ 000$	16	$\sqrt{56}$	5
0000 100	$s_\infty^* = 0011\ 001$	17	$\sqrt{61}$	4
0000 010				
0000 001				

The notion of p -norms for distance vectors is very common in many different research fields [33, 30, 21, 34, 20, 2, 27, 3, 17, 39]. In cluster analysis of data mining and machine learning, one main goal is to partition m observations (i.e., m real vectors of the same dimension) into K groups so that the sum of “discrepancies” between each observation and its nearest center is minimized. Here, two highly prominent clustering methods are K -means [32] and K -medians [25, 4] clustering, each using a slightly different notion of discrepancy measure. The first method aims to minimize the *sum of squared Euclidean distances* between each observation and the “mean” of its respective group. In other words, it minimizes the squared 2-norm of the Euclidean-distance vector. K -medians, on the other hand, uses the 1-norm instead of the squared 2-norm to define the discrepancy to the mean. Thus, instead of calculating the mean for each group to determine its centroid, one calculates the median.

In committee elections from social choice theory [17, 39, 35, 16], the p -norm is used to analyze how well a possible committee represents the voter's choices. In a fundamental approval-based procedure to select a t -person committee from n candidates, each voter either approves or disapproves each of the candidates, which can be expressed as a binary string of length n . An optimal committee is a length- n binary string containing exactly t ones and which minimizes the p -norm of the vector of the Hamming distances to each voter's preference string [39].

Problem Definition, Notation, and Conventions

Since the Hamming distance is frequently used in various applications, e.g., in computational biology [36], information theory, coding theory and cryptography [24, 11, 37], in social choice [26, 1] and since the notion of p -norm is very prominent in clustering tools [38, 6, 30, 40] and preference aggregation rules [1, 5, 35], where often $p = 1, 2, \infty$ but also other values of p are used, it is natural to consider computational problems associated with the p -norm of the Hamming distance metric. This is the main purpose of this paper. Specifically, we consider the following problem:

p -NORM HAMMING CENTROID (p -HDC)

Input: A set S of strings $s_1, \dots, s_m \in \{0, 1\}^n$ and a real k .

Question: Is there a string $s^* \in \{0, 1\}^n$ such that $\|(s^*, S)\|_p \leq k$?

Throughout, we will call a string s^* as above a *solution*. Note that there is nothing special about using the binary alphabet in the definition above, but for ease of presentation we use it throughout the paper. When $p = 1$, our p -HDC problem is precisely the CONSENSUS STRING problem, and when $p = \infty$ it becomes the CLOSEST STRING problem.

In the following, we list some notation and conventions that we use. By p -distance we mean the p^{th} -power of the Hamming distance. For each natural number t by $[t]$ we denote the set $\{1, 2, \dots, t\}$. Unless stated otherwise, by *strings* we mean binary strings over alphabet $\{0, 1\}$. Given a string s , we use $|s|$ to denote the length of this string. For two binary strings s and s' , let $s \circ s'$ denote the concatenation of s and s' . By $s[j]$ we denote the j th value or the value in the j^{th} character of string s . By $\bar{s} = (1 - s[j])_{j \in [|s|]}$ we denote the complement of the (binary) string s . Given two integers $j, j' \in \{1, 2, \dots, |s|\}$ with $j \leq j'$, we write $s|_j^{j'}$ for the substring $s[j]s[j+1] \cdots s[j']$. Given a number ℓ , we use $\mathbf{0}_\ell$ and $\mathbf{1}_\ell$ to denote the length- ℓ all-zero string and the length- ℓ all-one string, respectively.

Our Contributions

Our main result is a tight running time bound on the p -HDC problem for all fixed rationals $p > 1$. Specifically, we show that the problem is NP-hard and can be solved in $2^{k^{p/(p+1)+\varepsilon}} \cdot |I|^{O(1)}$ time for arbitrary small $\varepsilon > 0$ where $|I|$ denotes the size of the instance, but cannot be solved in $2^{o(k^{p/(p+1)})}$ time unless the Exponential Time Hypothesis (ETH) [12] fails. The lower bounds are given in Theorem 3 and Proposition 6 and the upper bound in Theorem 10. While the upper bound in this result is not very difficult, the lower bound uses an intricate construction and some delicate arguments to prove its correctness. In particular, the construction extensively utilizes the fact that since $p > 1$, the p -norm of Hamming distances is convex and always admits a second derivative. We believe that this kind of technique is of interest on its own. As another consequence of the hardness construction, we also obtain a $2^{o(n+m)}$ running time lower bound assuming ETH, which gives evidence that the trivial brute-force $2^n \cdot |I|$ -time algorithm for the problem cannot be substantially

improved. Moreover, the lower bounds also hold when we constrain the solution string to have a prescribed number of ones. That is, we also show hardness for the committee election problem mentioned above (Corollary 7).

In the final part of the paper we present two more algorithms for p -HDC. First, we provide an $m^{O(m^2)} \cdot |I|^{O(1)}$ time algorithm (see Theorem 13), by first formulating the problem as a so-called Combinatorial n -fold Integer Program, and then applying the algorithm developed by Knop et al. [28]. Second, we show that the problem can be approximated in polynomial time within a factor of 2, using an extension of the well known 2-approximation algorithm for CLOSEST STRING (see Proposition 14).

Related Work

The NP-complete CLOSEST STRING [18, 29] problem (aka. MINIMUM RADIUS) is a special case of p -HDC with $p = \infty$. It seems, however, difficult to adapt this hardness reduction to achieve our hardness results for every fixed rational p (see also the beginning of Section 2 for some more discussion). CLOSEST STRING has been studied extensively under the lens of parameterized complexity and approximation algorithmics. The first fixed-parameter algorithm for parameter k , the maximum Hamming distance bound, was given by Gram et al. [22], runs in $O(k^k \cdot km + mn)$ time where m and n denote the number and the length of input strings, respectively. This algorithm works for arbitrary alphabet Σ . For small alphabets Σ , there are algorithms with $O(mn + n \cdot |\Sigma|^{O(k)})$ running time [31, 9]. Both types of running time are tight under the ETH [12, Theorem 14.17]. For arbitrary alphabet Σ , Knop et al. [28] gave an algorithm with $m^{O(m^2)} \cdot \log n$ running time based on so-called n -fold integer programming. As for approximability, CLOSEST STRING admits a PTAS with running time $O(n^{O(\epsilon^{-2})})$ [31] but no EPTAS unless $\text{FPT} = \text{W}[1]$ [13].

Our problem falls into the general framework of convex optimization with binary variables. If the input and the output are allowed to have fractional values, then the underlying convex optimization problem, called L_p -NORM CONVEX MINIMIZATION, can be solved in polynomial time for each fixed value $p \leq 2$ [34, Chapter 6.3.2]. This convex optimization problem is a special variant of the in general NP-hard L_p SUBSPACE APPROXIMATION problem [14, 23]. This problem has as input m points s_1, \dots, s_m in \mathbb{R}^n and an integer k' , and asks to find a subspace H of \mathbb{R}^n of dimension k' that minimizes the following $\sum_{i=1}^m (\text{dist}^p(H, a_i))^{1/p}$, where $\text{dist}(H, a_i)$ is the minimum Euclidean distance between a_i and any point in H . For $k = 0$, L_p SUBSPACE APPROXIMATION is equivalent to the L_p -NORM CONVEX MINIMIZATION.

For $p \in \{2, 3\}$, maximizing (instead of minimizing) the p -norm reduces to MIRKIN MINIMIZATION in consensus clustering with input and output restricted to two-clusters, which was shown to be NP-hard [15] under Turing reductions. Recently, Chen et al. [7] showed that the simple 2^n -time algorithm by brute-force searching all possible outcome solutions is essentially tight under ETH. They also provided some efficient algorithms and showed that the problem admits an FPTAS using a simple rounding technique.

2 NP-hardness for the p -Norm of Hamming Distance Vectors

We now show that p -HDC is NP-hard for each fixed rational number $p > 1$ (Theorem 3 and Proposition 6) and that algorithms with running time $2^{o(n+m)}$ or $2^{o(k^{p/(p+1)})}$ would contradict the ETH. We reduce from the NP-hard 3-COLORING problem [19] in which, given an undirected graph $G = (V, E)$, we ask whether there is a *proper vertex coloring* $\text{col}: V \rightarrow \{0, 1, 2\}$, that is, no two *adjacent* vertices receive the same color.

The first challenge we need to overcome when designing the reduction is to produce some regularity in the solution string: Given $\hat{n} \in \mathbb{N}$, in Lemma 1, we show how to construct a set of strings to enforce a solution string to have exactly \hat{n} ones which only occur in the columns of some specific range. This allows us later on to build gadgets that have several discrete states. Indeed, after controlling the overall number of ones in the solution in this way, we can allocate three columns (one for each color) for each vertex v in G and build a gadget for v such that this gadget induces minimum p -distance to the solution if and only if there is exactly 1 one in the solution in the columns allocated for v . This column determines the color for v . Then, for each edge, we will introduce an edge gadget consisting of six strings which induce minimum p -distance in the solution if and only if they are “covered” by the ones in the solution exactly twice, corresponding to different colors.

In general, the design of gadgets for p -HDC is quite different from the known NP-hard case CLOSEST STRING ($p = \infty$) [18, 29]: In CLOSEST STRING every optimal solution s^* must regard the “worst” possible input string while in our case s^* can escape such constraints by distributing some of its Hamming distance from the “worst” to other strings.

In the remainder of this section, let a and b be two fixed integers such that a and b are coprime, $a > b$, and $p = a/b > 1$. To better capture the Hamming distance, we introduce the notion of the *Hamming set* of two strings s and s' of equal length n , which consists of the indices of the columns at which both strings differ: $\text{hs}(s, s') = \{j \in [n] \mid s[j] \neq s'[j]\}$.

As mentioned, we first show how to construct a set of strings to enforce some structure on the optimal solution, that is, a binary string with minimum sum of the p -distances.

► **Lemma 1** (\star^1). *Let $p > 1$ be a fixed rational number, and a and b be two coprime fixed integers with $p = a/b$. Let S consist of one string $\mathbf{1}_{(2^b+1)\hat{n}}\mathbf{0}_{\hat{n}}$ and 2^{a-b} copies of string $\mathbf{0}_{(2^b+2)\hat{n}}$, where \hat{n} is a positive integer. For each string $s^* \in \{0, 1\}^{(2^b+2)\hat{n}}$, the following holds.*

- (1) *If $d(s^*, \mathbf{0}_{(2^b+2)\hat{n}}) = \hat{n}$ and $\text{hs}(s^*, \mathbf{0}_{(2^b+2)\hat{n}}) \subseteq [(2^b+1)\hat{n}]$, then $\|(s^*, S)\|_p^p = (2^a + 2^{a-b}) \cdot \hat{n}^p$.*
- (2) *If $d(s^*, \mathbf{0}_{(2^b+2)\hat{n}}) \neq \hat{n}$ or $\text{hs}(s^*, \mathbf{0}_{(2^b+2)\hat{n}}) \not\subseteq [(2^b+1)\hat{n}]$, then $\|(s^*, S)\|_p^p > (2^a + 2^{a-b}) \cdot \hat{n}^p$.*

To show Lemma 1 we crucially use the fact that $p > 1$. In contrast, if $p = 1$, then taking the majority value in each column yields an optimal solution, and thus it is impossible to force every optimal solution to have a certain number of ones without at the same time specifying in which precise columns these ones should occur.

In the reduction we make heavy use of specific pairs of strings whose Hamming distances to an arbitrary string always sum up to some lower bound. They will enforce local structure in some columns of the solution, while being somewhat immune to changes elsewhere. As a tool in the reduction we derive the following lower bound on the p -distance of an arbitrary string to a pair of strings which are quite far from each other, in terms of Hamming distances.

► **Lemma 2** (\star). *Let s_1 and s_2 be two strings of the same length R such that the Hamming distance between s_1 and s_2 is $d(s_1, s_2) = 2L$. For each rational $p > 1$ and each length- R string \hat{s} the following holds.*

- (1) $d^p(\hat{s}, s_1) + d^p(\hat{s}, s_2) \geq 2 \cdot L^p$.
- (2) *If $d(\hat{s}, s_1) = d(\hat{s}, s_2) = L$, then $d^p(\hat{s}, s_1) + d^p(\hat{s}, s_2) = 2 \cdot L^p$.*
- (3) *If $d(\hat{s}, s_1) \neq L$ or $d(\hat{s}, s_2) \neq L$, then $d^p(\hat{s}, s_1) + d^p(\hat{s}, s_2) > 2 \cdot L^p$.*

Using Lemmas 1 and 2, we can show NP-hardness of p -HDC for each fixed rational $p > 1$. For better readability, we will first show hardness for the case with multiple identical strings (Theorem 3) and then extend the construction to also include the case where no two strings are the same (Proposition 6).

¹ Proofs for results marked by \star can be found in [8].

► **Theorem 3.** For each fixed rational number $p > 1$, p -HDC (with possibly multiple identical strings) is NP-hard.

Proof. First of all, let a and b be two fixed coprime integers such that $p = a/b$. To show the hardness result, we reduce from the NP-hard 3-COLORING problem [19] defined above. Let $G = (V, E)$ be an instance of 3-COLORING. Let n be the number of vertices in G and m the number of edges. Denote $V = \{v_1, v_2, \dots, v_n\}$ and $E = \{e_1, e_2, \dots, e_m\}$.

Construction. We introduce three groups of strings of length $(2^b + 2) \cdot \hat{n}$ each, where $\hat{n} = n + m$. The first group ensures that each optimal solution string must have exactly \hat{n} ones which appear in the first $3\hat{n}$ columns (using Lemma 1), the second group ensures that an optimal solution enforces that each vertex has exactly one of the three colors, and the third group, combined with the second group, ensures that no two adjacent vertices obtain the same color.

Group 1. Construct one string $\mathbf{1}_{(2^b+1)\hat{n}} \circ \mathbf{0}_{\hat{n}}$ and 2^{a-b} copies of the same string $\mathbf{0}_{(2^b+2)\hat{n}}$.

Group 2. This group consists of one pair of strings for each vertex. Each pair consists of two strings which are mostly complements to each other. This ensures that the Hamming distance to the solution induced by a pair is somewhat homogeneous, regardless where exactly the ones in the solution occur. However, in each pair there are three columns, corresponding to the vertex, which will skew the pairs of Hamming distances in a way to induce minimum p -distances only if the solution has exactly 1 one in these three columns. Formally, for each vertex $v_i \in V$, let u_i be a string of length $3\hat{n}$ which has exactly 3 ones in the columns $3i - 2, 3i - 1, 3i$, and let \bar{u}_i be the complement of u_i . Deriving from u_i , we construct two *vertex strings* s_i and r_i with $s_i := u_i \circ \mathbf{0}_{(2^b-2)\hat{n}} \circ \mathbf{0} \circ \mathbf{1}_{\hat{n}-1}$ and $r_i := \bar{u}_i \circ \mathbf{0}_{(2^b-2)\hat{n}} \circ \mathbf{1} \circ \mathbf{0}_{\hat{n}-1}$. Note that both strings s_i and r_i have all zeros in the columns $\{3\hat{n}, \dots, (2^b + 1)\hat{n}\}$ such that $d(s_i, r_i) = 4\hat{n}$.

For an illustration, the strings s_2 and r_2 , which correspond to the vertex v_2 , are as follows:
 $s_2 = \mathbf{000111} \circ \mathbf{0}_{3\hat{n}-6} \circ \mathbf{0}_{(2^b-2)\hat{n}} \circ \mathbf{0} \circ \mathbf{1}_{\hat{n}-1}$, $r_2 = \mathbf{111000} \circ \mathbf{1}_{3\hat{n}-6} \circ \mathbf{0}_{(2^b-2)\hat{n}} \circ \mathbf{1} \circ \mathbf{0}_{\hat{n}-1}$.

Group 3. We now use three pairs of strings for each edge to ensure relatively homogeneous distributions of Hamming distances to the solution and then skew them. This time, we aim to skew distances to the solution so that their corresponding p -distances are minimum only if the solution distributes exactly three ones (corresponding to the colors) over three special regions: two corresponding to the endpoints of the edge and one extra dummy region.

Formally, for each edge $e_j \in E$ let $e_j^{(0)}, e_j^{(1)}$, and $e_j^{(2)}$ denote three strings, each of length $3\hat{n}$, that ensure that the edge and both of its endpoints each have a distinct color:

$$\forall \ell \in \{1, 2, \dots, \hat{n}\}: e_j^{(0)}[3\ell - 2, 3\ell - 1, 3\ell] := \begin{cases} 100, & 1 \leq \ell \leq n \text{ with } v_\ell \in e_j, \text{ or } \ell = j + n, \\ 000, & \text{otherwise.} \end{cases}$$

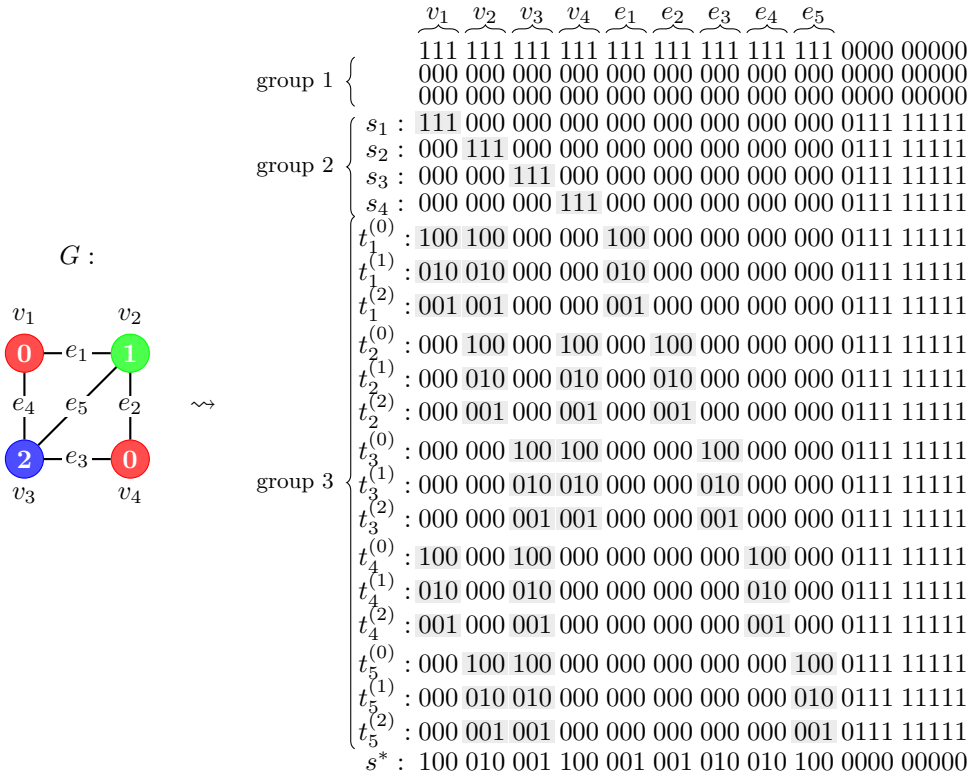
$$e_j^{(1)}[3\ell - 2, 3\ell - 1, 3\ell] := \begin{cases} 010, & 1 \leq \ell \leq n \text{ with } v_\ell \in e_j, \text{ or } \ell = j + n, \\ 000, & \text{otherwise.} \end{cases}$$

$$e_j^{(2)}[3\ell - 2, 3\ell - 1, 3\ell] := \begin{cases} 001, & 1 \leq \ell \leq n \text{ with } v_\ell \in e_j, \text{ or } \ell = j + n, \\ 000, & \text{otherwise.} \end{cases}$$

Now, we construct the following six *edge strings* for edge e_j :

$$\forall z \in \{0, 1, 2\}: t_j^{(z)} := e_j^{(z)} \circ \mathbf{0}_{(2^b-2)\hat{n}} \circ \mathbf{0} \circ \mathbf{1}_{\hat{n}-1} \text{ and } w_j^{(z)} := \bar{e}_j^{(z)} \circ \mathbf{0}_{(2^b-2)\hat{n}} \circ \mathbf{1} \circ \mathbf{0}_{\hat{n}-1}.$$

Just as for group 2, the two strings $t_j^{(z)}$ and $w_j^{(z)}$ have all zeros in the columns $\{3\hat{n}, \dots, (2^b + 1)\hat{n}\}$ such that $d(t_j^{(z)}, w_j^{(z)}) = 4\hat{n}$.



■ **Figure 1** Illustration of the reduction used in Theorem 3. The left figure depicts a graph G that admits a proper vertex coloring col (see the labels on the vertices). For instance, vertex v_1 has color 0, i.e., $\text{col}(v_1) = 0$. The right figure shows the crucial part of an instance of p -HDC with $p = 2$ (i.e., $a = 2$ and $b = 1$) that we will construct according to the proof for Theorem 3. For each pair of constructed strings we only show the first one. A solution string s^* corresponding to the coloring col is depicted at the bottom of the right figure.

For an example, assume that $a = 3$, $b = 2$, $n = 3$, and $m = 2$, and there is an edge of the form $e_2 = \{v_1, v_3\}$. Then, the two triples of strings that we construct for e_2 have each length $(2^b + 2)(n + m) = 30$ and are

$$\begin{aligned}
 t_j^{(0)} &= 100\ 000\ 100\ 000\ 100\ 0000000000\ 01111, & w_j^{(0)} &= 011\ 111\ 011\ 111\ 011\ 0000000000\ 10000, \\
 t_j^{(1)} &= 010\ 000\ 010\ 000\ 010\ 0000000000\ 01111, & w_j^{(1)} &= 101\ 111\ 101\ 111\ 101\ 0000000000\ 10000, \\
 t_j^{(2)} &= 001\ 000\ 001\ 000\ 001\ 0000000000\ 01111, & w_j^{(2)} &= 110\ 111\ 110\ 111\ 110\ 0000000000\ 10000.
 \end{aligned}$$

Summarizing, the instance I' of p -HDC consists of the following strings, each of length $(2^b + 2)\hat{n} = (2^b + 2)(n + m)$:

- (1) Add the $2^{a-b} + 1$ strings in group 1 to I' .
- (2) For each vertex $v_i \in V$, add the vertex strings s_i and r_i to I' .
- (3) For each edge $e_j \in E$, add two triples $t_j^{(0)}, t_j^{(1)}, t_j^{(2)}$ and $w_j^{(0)}, w_j^{(1)}, w_j^{(2)}$ to I' .

See Figure 1 for an illustration.

Finally, we define k such that $k^p = (2^a + 2^{a-b}) \cdot \hat{n}^p + 2(n + 3m) \cdot (2\hat{n})^p$. This completes the construction, which can clearly be done in polynomial time.

Correctness of the construction. Before we show the correctness of our construction, we define a notion and make an observation. Let s and s' be two strings of equal length. We say that s covers s' exactly once if there is exactly one integer $\ell \in \{1, 2, \dots, |s|\}$ with $s[\ell] = s'[\ell] = 1$.

▷ Claim 4 (★). Let s^* and s be two strings, both of length $4\hat{n}$, such that

(i) s^* has exactly \hat{n} ones and each of them is in the first $3\hat{n}$ columns, and

(ii) in s , the first $3\hat{n}$ columns have exactly 3 ones and the last \hat{n} columns are $0 \circ \mathbf{1}_{\hat{n}-1}$.

Then, if s^* covers s exactly once, then $d^p(s^*, s) + d^p(s^*, \bar{s}) = 2 \cdot (2\hat{n})^p$; else $d^p(s^*, s) + d^p(s^*, \bar{s}) > 2 \cdot (2\hat{n})^p$.

We show that G has a proper 3-coloring if and only if there is a string s^* such that the sum of the p -distances from s^* to all strings in I' is at most $k^p = (2^a + 2^{a-b}) \cdot \hat{n}^p + 2(n+3m) \cdot (2\hat{n})^p$.

For the “if” direction, let s^* be a string which has a sum of p -distances of at most k^p to all strings in I' . Before we define a coloring for the vertices and show that it is proper we observe several properties of the solution string s^* .

By Lemma 2(1), the sum of p -distances to all strings from group 2 and group 3 is at least $2 \cdot (2\hat{n})^p \cdot (n+3m)$ since these groups consist of $n+3m$ pairs of strings, and the strings in each of these pairs have Hamming distance exactly $4\hat{n}$ to each other. By the definition of k , the sum of p -distances from s^* to the first of group of strings is thus at most $(2^a + 2^{a-b}) \cdot \hat{n}^p$. Hence, by the contra-positive of Lemma 1(2), the solution string s^* has exactly \hat{n} ones, which all appear in the first $(2^b + 1)\hat{n}$ columns, i.e., $d(s^*, \mathbf{0}_{(2^b+2)\hat{n}}) = \hat{n}$ and $\text{hs}(s^*, \mathbf{0}_{(2^b+2)\hat{n}}) \subseteq [(2^b + 1)\hat{n}]$. By Lemma 1(1), this implies that

$$\sum_{s \in \text{group 1}} d^p(s^*, s) = (2^a + 2^{a-b}) \cdot \hat{n}^p. \quad (1)$$

Next, we claim that the ones in the solution s^* indeed all appear in the first $3\hat{n}$ columns, i.e., $\text{hs}(s^*, \mathbf{0}_{(2^b+2)\hat{n}}) \subseteq [3\hat{n}]$. Suppose, for the sake of contradiction, that solution s^* contains x ones which appear in columns ranging from $3\hat{n} + 1$ to $(2^b + 1)\hat{n}$ with $x > 0$. Consider an arbitrary pair of strings s_i and r_i from group 2 or an arbitrary pair of strings $t_i^{(z)}$ and $w_i^{(z)}$ from group 3; for the sake of readability, represent them by s and s' . By construction, strings s and s' have Hamming distance exactly $4\hat{n}$ to each other, but have all zeros in the columns between $3\hat{n} + 1$ and $(2^b + 1)\hat{n}$. Since $x > 0$, by the triangle inequality of Hamming distances, it follows that at least one string from the pair, s or s' , has Hamming distance more than $2\hat{n}$ from s^* . However, by Lemma 2(3), this means that the sum of p -distances from s^* to $\{s, s'\}$ exceeds $2 \cdot (2\hat{n})^p$. Since there are in total $n + 3m$ such pairs in groups 2 and 3, the sum of p -distances from s^* to these groups exceeds $2(n + 3m) \cdot (2\hat{n})^p$, a contradiction to equation (1) and the defined bound k . Thus, indeed, it holds that

$$d(s^*, \mathbf{0}_{(2^b+2)\hat{n}}) = \hat{n} \text{ and } \text{hs}(s^*, \mathbf{0}_{(2^b+2)\hat{n}}) \subseteq [3\hat{n}]. \quad (2)$$

This implies that, when determining the p -distance of s^* to the strings from group 2 and group 3, we can ignore, the values in the columns ranging from $3\hat{n} + 1$ to $(2^b + 1)\hat{n}$, in each string, which includes the solution s^* , because s^* also has only zeros in these columns. We will hence from now on treat these columns as if they do not exist. In this way, we obtain strings of length $4\hat{n}$. Again, consider an arbitrary pair of strings s_i and r_i from group 2 (resp. an arbitrary pair of strings $t_i^{(z)}$ and $w_i^{(z)}$ from group 3), and represent them by s and s' . Since we ignore columns $3\hat{n} + 1$ to $(2^b + 1)\hat{n}$, string s' is the complement of s . By construction, the Hamming distance between s and s' is exactly $4\hat{n}$. Using Claim 4 on s^* , s, s' , the sum of p -distances from s^* to the pair $\{s, s'\}$ is indeed equal to $2 \cdot (2\hat{n})^p$. By the same claim, it follows that s^* covers each string s_i (resp. $t_j^{(z)}$) from group 2 (resp. group 3) exactly once.

Having this property, we are ready to color the vertices. Let $\text{col}: V \rightarrow \{0, 1, 2\}$ be a mapping defined as follows. For each $v_i \in V$, set $\text{col}(v_i) = z$ where $z \in \{0, 1, 2\}$ such that $s^*[3i - 2 + z] = 1$. Note that, since s^* covers s_i exactly once and since s_i has exactly three

ones in the columns $3i - 2$, $3i - 1$, and $3i$, there is indeed such a z with $\text{col}(v_i)$. We claim that col is a proper coloring for G . Suppose, towards a contradiction, that there is an edge $e_j = \{v_i, v_{i'}\} \in E$ such that v_i and $v_{i'}$ have the same color from col , say $z \in \{0, 1, 2\}$. By the definition of col , this means that $s^*[3i - 2 + z] = s^*[3i' - 2 + z] = 1$. However, by the definition of the string $t_j^{(z)}$ which corresponds to the edge e_j , we also have that $t_j^{(z)}[3i - 2 + z] = t_j^{(z)}[3i' - 2 + z] = 1$. This implies that $t_j^{(z)}$ is not covered by s^* exactly once – a contradiction to our reasoning above that s^* covers each string from the third group exactly once.

For the “only if” direction, let $\text{col}: V \rightarrow \{0, 1, 2\}$ be a proper coloring for G . For an edge $e \in E$ with two endpoints $v_i, v_{i'}$, let $\text{col}(e) = \{\text{col}(v_i), \text{col}(v_{i'})\}$. We claim that string s^* , defined as follows, has the desired bound on the sum of the p -distances to all strings of I' .

$$\forall i \in \{1, 2, \dots, n\}: s^*[3i - 2, 3i - 1, 3i] := \begin{cases} 100, & \text{col}(v_i) = 0, \\ 010, & \text{col}(v_i) = 1, \\ 001, & \text{col}(v_i) = 2. \end{cases}$$

$$\forall j \in \{n + 1, n + 2, \dots, \hat{n}\}: s^*[3j - 2, 3j - 1, 3j] := \begin{cases} 100, & \text{col}(e_j) = \{1, 2\}, \\ 010, & \text{col}(e_j) = \{0, 2\}, \\ 001, & \text{col}(e_j) = \{0, 1\}. \end{cases}$$

$$s^* \Big|_{3\hat{n}+1}^{(2^b+2)\hat{n}} := \mathbf{0}_{\hat{n}}.$$

First of all, since col is a proper coloring, s^* is well defined in all $(2^b + 2)\hat{n}$ columns. Moreover, it has exactly n ones in the first $3n$ columns and exactly m ones in the next $3m$ columns, and all zeros in the remaining columns. Thus, by Lemma 1(2), the sum of the p -distances from s^* to the first group of strings is $(2^a + 2^{a-b}) \cdot \hat{n}^p$.

Now, we focus on strings from group 2 and group 3. Since the solution s^* and each string in these groups have only zeros in the columns between $3\hat{n} + 1$ and $(2^b + 1)\hat{n}$, we can simply ignore the values in these columns and assume from now on that the strings have length $4\hat{n}$. Moreover, for each $i \in [n]$, the pair s_i and r_i can be considered as complement to each other. Thus, for each string s_i from group 2, s^* and s_i fulfill the properties stated in Claim 4. Moreover, by definition, s^* covers s_i exactly once. Thus, by the same claim, we have that the sum of the p -distances from s^* to all strings in group 2 is $n \cdot 2 \cdot (2\hat{n})^p$.

Analogously, consider a string $t_j^{(z)}$ from group 3, $j \in \{1, 2, \dots, m\}$ and $z \in \{0, 1, 2\}$. Recall that $t_j^{(z)}$ corresponds to the edge e_j , and let v_i and $v_{i'}$ be the two endpoints of edge e_j . We claim that s^* covers $t_j^{(z)}$ exactly once. Observe that $t_j^{(z)}$ has exactly 3 ones in the first $3\hat{n}$ columns, namely at columns $3i - 2 + z$, $3i' - 2 + z$, and $3n + 3j - 2 + z$. To prove that s^* covers $t_j^{(z)}$ exactly once, it suffices to show that s^* has 1 one in exactly one of these three columns. To show this, we consider the substrings $t_j^{(z)} \Big|_{3n+3j-2}^{3n+3j}$ and $s^* \Big|_{3n+3j-2}^{3n+3j}$.

Case 1: $s^* \Big|_{3n+3j-2}^{3n+3j} = t_j^{(z)} \Big|_{3n+3j-2}^{3n+3j}$. By the definition of s^* , this implies that $s^*[3n + 3j - 2 + z] = 1$ and $\text{col}(e_j) = \{0, 1, 2\} \setminus \{z\}$. We claim that $s^*[3i - 2 + z] = s^*[3i' - 2 + z] = 0$. By the definition of s^* regarding the columns that correspond to the endpoint v_i of edge e_j , we have that $s^*[3i - 2 + \text{col}(v_i)] = 1$ while $s^*[3i - 2 + z] = 0$ (since $z \notin \text{col}(e_j) = \{\text{col}(v_i), \text{col}(v_{i'})\}$). Analogously, by the definition of s^* regarding the columns that correspond to the other endpoint $v_{i'}$ of edge e_j , we have that $s^*[3i' - 2 + \text{col}(v_{i'})] = 1$ while $s^*[3i' - 2 + z] = 0$ (since $z \notin \text{col}(e_j) = \{\text{col}(v_i), \text{col}(v_{i'})\}$). Thus, $3n + 3j - z$ is the only column in which both s^* and $t_j^{(z)}$ have 1 one, implying that s^* covers $t_j^{(z)}$ exactly once.

Case 2: $s^*|_{3n+3j-2}^{3n+3j} \neq t_j^{(z)}|_{3n+3j-2}^{3n+3j}$. This means that $s^*[3n+3j-2+z] = 0$ and that $z \in \text{col}(e_j)$. To show that s^* covers $t_j^{(z)}$ exactly once in this case, it suffices to show that either $s^*[3i-2+z] = 1$ and $s^*[3i'-2+z] = 0$, or $s^*[3i-2+z] = 0$ and $s^*[3i'-2+z] = 1$.

- Assume that $s^*[3i-2+z] = 1$. Then, by the definition of s^* regarding the columns that correspond to the endpoint v_i of edge e_j , this means that $\text{col}(v_i) = z$. Since col is a proper coloring, it follows that $\text{col}(v_{i'}) \neq z$. Thus, again by the definition of s^* regarding the columns that correspond to the other endpoint $v_{i'}$ of edge e_j , it follows that $s^*[3i'-2+z] = 0$.
- Assume that $s^*[3i-2+z] = 0$. Then, by the definition of s^* regarding the columns that correspond to the endpoint v_i of edge e_j , we have $\text{col}(v_i) \neq z$. Since $z \in \text{col}(e_j)$ and col is a proper coloring, the other endpoint $v_{i'}$ of edge e_j must have color $\text{col}(v_{i'}) = z$. Again, by the definition of s^* regarding the columns that correspond $v_{i'}$, it follows that $s^*[3i'-2+z] = 1$.

We have just shown that s^* covers $t_j^{(z)}$ exactly once. Since s^* and $t_j^{(z)}$ fulfill the property stated in Claim 4, it follows from the same claim that the sum of p -distances from s^* to $t_j^{(z)}$ and to $w_j^{(z)}$ is $2 \cdot (2\hat{n})^p$. There are $3m$ pairs in this group. So, the sum of the p -distances from s^* to all strings of this group is $3m \cdot 2 \cdot (2\hat{n})^p$.

In total, the sum of the p -distances from s^* to all strings of I' is $(2^a + 2^{a-b}) \cdot \hat{n}^p + 2 \cdot (2\hat{n})^p \cdot (n + 3m) = k^p$, as required. ◀

Our NP-hardness reduction implies the following running time lower bounds [12].

► **Corollary 5** (★). *For each fixed rational number $p > 1$, unless the ETH fails, no $2^{o(\hat{n}+\hat{m})} \cdot |I'|^{O(1)}$ -time or $2^{o(k^{p/(p+1)})} \cdot |I'|^{O(1)}$ -time algorithm exists that decides every given instance I' of p -HDC where \hat{n} is the length of the input strings, \hat{m} is the number of input strings, and k is the p -norm bound.*

Using a slight modification of the construction, we can show that our results are not idiosyncratic to instances which contain some strings multiple times. (Recall that the gadget from Lemma 1 in the construction contains 2^{a-b} copies of the all-zero string.) The basic idea is to append an identity matrix to the strings we need to distinguish, and then to show using a slightly more involved analysis that the gadgets still work in the same way.

► **Proposition 6** (★). *Theorem 3 and Corollary 5 hold even if all input strings are distinct.*

Let p -NORM APPROVAL COMMITTEE be the variant of p -HDC in which we additionally get $t \in \mathbb{N}$ as an input and require the number of ones in the solution string s^* to be exactly t [39]. Note that in the proof of Theorem 3 we have first shown that each solution string contains exactly \hat{n} ones. Thus, the reduction works in the same way for p -NORM APPROVAL COMMITTEE when we specify $t = \hat{n}$ in the constructed instance. We hence obtain the following.

► **Corollary 7**. *For each fixed rational $p > 1$, p -NORM APPROVAL COMMITTEE is NP-hard and admits no algorithm running in $2^{o(\hat{n}+\hat{m})} \cdot |I'|^{O(1)}$ -time or in $2^{o(k^{p/(p+1)})} \cdot |I'|^{O(1)}$ -time unless the ETH fails, where \hat{n} is the number of candidates, \hat{m} is the number of voters, and k is the p -norm bound.*

3 Algorithmic Results

We now turn to our positive results. In Section 3.1 we provide an efficient algorithm when the objective value k is small. In Section 3.2, we derive an integer convex programming formulation to obtain an efficient algorithm for instances where the number m of input strings is small. Finally, we give a simple 2-approximation in Section 3.3.

3.1 A Subexponential-Time Algorithm

In this section, we present an algorithm with running time $2^{k^{p/(p+1)+\epsilon}} \cdot |I|^{O(1)}$ for any $\epsilon > 0$ and input instance I with distance bound k . By the lower bound result given in Corollary 5, we know that under ETH, the running time of the obtained algorithm is tight.

The algorithm is built on two subcases, distinguishing on a relation between the number m of input strings and the distance bound k . In each subcase we use a distinct algorithm that runs in subexponential time when restricted to that subcase. To start with, a dynamic programming algorithm which keeps track of the achievable vector of Hamming distances to each input string after columns 1 to $j \leq n$ has running time $O(n \cdot k^m)$.

► **Lemma 8** (\star). *p -HDC can be solved in $O(n \cdot k^m)$ time and space, where m and n are the number and the length of the input strings, respectively, and k is the p -norm distance bound.*

The dynamic program given in Lemma 8 is efficient if there is a small number m of input strings only. In particular, if m satisfies $m \leq \frac{k^{p/(p+1)}}{\log k}$, then we immediately obtain an $O(n \cdot 2^{k^{p/(p+1)}})$ -time algorithm. Otherwise, we can use Lemma 9. The algorithm behind Lemma 9 is based on a different but related idea as the fixed-parameter algorithm for CLOSEST STRING given by Gramm et al. [22]: We use data reduction to shrink the length of the strings by k^p , observe that one of the input strings must be close to a solution with bound k if it exists, and then find the solution by a search tree.

► **Lemma 9**. *p -HDC can be solved in $O(nm^2 \cdot k^{\frac{p \cdot k}{\sqrt[p]{m}}})$ time, where m and n are the number and the length of the input strings, respectively, and k is the p -norm distance bound.*

Proof. Let $I = (S, k)$ be an instance of p -HDC with $S = (s_1, \dots, s_m)$ being the input strings of length n and k being the p -norm distance bound. To show the statement, we first observe that if a column is an all-zero (resp. an all-one) column, then we can simply assume that an optimal solution will also have zero (resp. one) in this column as our objective function is convex. By preprocessing all columns that are either an all-zero or an all-one vector, we obtain an equivalent instance, where each column has at least a zero and at least a one. Thus, for each column, no matter which value a solution has at this column, it will always induce Hamming distance of at least one to some input string. Consequently, if there are more than k^p columns remaining, then we can simply answer “no” as any string will have cost more than k to the input. Otherwise, there remain at most k^p columns.

If I is a yes-instance, meaning that there is a solution s^* for I with $\|(s^*, S)\|_p \leq k$, then there is an input string $s^{**} \in S$ whose Hamming distance satisfies $d(s^{**}, s^*) \leq \sqrt[p]{\frac{k^p}{m}} = \frac{k}{\sqrt[p]{m}}$. Thus, we iterate over all input strings in S , assuming in each iteration that the current string is the aforementioned s^{**} . For each string s_i that we assume to be the aforementioned s^{**} , we go over all strings \hat{s} that differ from s_i by k' columns with $k' \leq \frac{k}{\sqrt[p]{m}}$. We check whether $\|(\hat{s}, S)\|_p \leq k$. We answer “no” if for each input string $s_i \in S$, no length- n string \hat{s} with $d(s_i, \hat{s}) \leq \frac{k}{\sqrt[p]{m}}$ exists which satisfies $\|(\hat{s}, S)\|_p \leq k$.

It remains to show the running-time bound. Observe that the preprocessing for all-zero and all-one columns can be done in $O(nm)$ time. After that, for each of the m input strings s_i , we search all strings of Hamming distance at most $k' \leq \frac{k}{\sqrt[p]{m}}$ to s_i , and there are $O(n^{\frac{k}{\sqrt[p]{m}}})$ such strings. For each of them, we compute the objective function, which can be accomplished in $O(nm)$ time. As already reasoned, after the preprocessing, n is upper-bounded by k^p . Thus, the overall running time bound is $O(nm + nm^2 \cdot n^{\frac{k}{\sqrt[p]{m}}}) = O(nm^2 \cdot k^{\frac{p \cdot k}{\sqrt[p]{m}}})$, as claimed. ◀

Combining Lemma 8 with Lemma 9, we obtain a subexponential algorithm with respect to k .

► **Theorem 10.** *For each fixed positive value $\varepsilon > 0$, p -HDC can be solved in $O(nm^2 \cdot 2^{k^{p/(p+1)+\varepsilon}})$ time, where n and m denote the length and the number of input strings, and k is the p -norm distance bound with $p > 1$.*

Proof. Let $I = (S, k)$ be an instance of p -HDC with $S = (s_1, \dots, s_m)$ being the input strings of length n and k being the p -norm distance bound. As already discussed, to solve our problem we distinguish between two cases, depending on whether $m \leq \frac{k^{p/(p+1)}}{\log k}$ holds.

If $m \leq \frac{k^{p/(p+1)}}{\log k}$, then $k^m \leq k^{\frac{k^{p/(p+1)}}{\log k}} \leq 2^{k^{p/(p+1)}}$. In this case, we use the dynamic programming approach given in the proof of Lemma 8, which has the desired running time $O(n \cdot k^m) = O(n \cdot 2^{k^{p/(p+1)}})$.

Otherwise, $m > \frac{k^{p/(p+1)}}{\log k}$, meaning that $\frac{p \cdot k \cdot \log k}{\sqrt[p]{m}} < p \cdot k \cdot \log k / \sqrt[p]{\frac{k^{p/(p+1)}}{\log k}} = p \cdot k^{p/(p+1)} \cdot (\log k)^{(p+1)/p}$. For each fixed positive $\varepsilon \in \mathbb{R}$ there exists $k_0 = k_0(p, \varepsilon) \in \mathbb{R}$ such that, for each $k \geq k_0$, we have $p \cdot (\log k)^{(p+1)/p} < k^\varepsilon$. If $k < k_0$, then the algorithm in the proof of Lemma 9 runs in $O(nm^2)$ time. Otherwise $k \geq k_0$, which implies $\frac{p \cdot k \cdot \log k}{\sqrt[p]{m}} < k^{p/(p+1)+\varepsilon}$. Thus, the algorithm given in the proof of Lemma 9 has a running time of $O(nm^2 \cdot k^{\frac{p \cdot k}{\sqrt[p]{m}}}) = O(nm^2 \cdot 2^{\frac{p \cdot k \cdot \log k}{\sqrt[p]{m}}}) = O(nm^2 \cdot 2^{k^{p/(p+1)+\varepsilon}})$.

Altogether we presented an algorithm which has the desired running time bound. ◀

3.2 A Fixed-Parameter Algorithm for the Number of Input Strings

In this section, we show that minimizing the sum of the p -distances is fixed-parameter tractable for the number m of input strings. The idea is to formulate our problem as a combinatorial n -fold integer program (CnIP) with $O(2^m)$ variables and $O(m)$ constraints. We then apply the following simplified result of Knop et al. [28]:

► **Proposition 11** ([28, Theorem 3]). *Let $E \in \mathbb{Z}^{(r+1) \times t}$ be a matrix such that the last row equals $(1, 1, \dots, 1) \in \mathbb{Z}^t$. Let $b \in \mathbb{Z}^{r+1}$, $\ell, u \in \mathbb{Z}^t$, and let $f: \mathbb{R}^t \rightarrow \mathbb{R}$ be a separable convex function given by an evaluation oracle². Then, there is an algorithm that solves³ $P := \min\{f(x) \mid Ex = b \wedge \ell \leq x \leq u \wedge x \in \mathbb{Z}^t\}$ in $t^{O(r)} \cdot ((1 + \|E\|_\infty) \cdot r)^{O(r^2)} \cdot L + T$ time, where L is the total bit-length of b, ℓ, u , and the oracle of f , and T is the time that an algorithm needs to solve the continuous relaxation of P .*

To get a useful running time bound from Proposition 11, we need a bounded number of variables. To do this, we group columns in the input strings with the same “type” together and introduce an integer variable for each column type. To this end, given a set $S = \{s_1, \dots, s_m\}$ of length- n strings, we say that two columns $j, j' \in [n]$ have the *same type* if for each $i \in [m]$ it holds that $s_i[j] = s_i[j']$. The *type* of column j is its equivalence class in the same-type relation. Thus, each type is represented by a vector in $\{0, 1\}^m$. Let n' denote the number of different (column) types in S . Then, $n' \leq \min(2^m, n)$. Enumerate the n' column types as $t_1, \dots, t_{n'}$. Below we identify a column type with its index for easier notation. Using this, we can encode the set S succinctly by introducing a constant $e(j)$ for each column type $j \in [n']$ that denotes the number of columns with type j . Given a solution string s^* , we can also encode this string s^* via an integer vector $x \in \{0, 1, \dots, n\}^{n'}$, where for each type $j \in [n']$ we define $x[j]$ as the number of ones in the solution s^* whose corresponding columns are of type j . Note that this encodes all essential information in a solution, since the actual

² A function is separable convex if it is the sum of univariate convex functions.

³ The algorithm correctly reports either a minimizer $x \in P$ or that P is infeasible or unbounded.

order of the columns is not important (see Example 12). Vice versa, each integer vector in $x \in \{0, 1, \dots, n\}^{n'}$ satisfying $0 \leq x[j] \leq e(j)$ for each $j \in [n']$ yields a length- n binary string $s^*(x)$; it remains to add constraints and a suitable objective function to ensure that $s^*(x)$ has minimum sum of p -distances to the input strings.

► **Example 12.** For an illustration, let $S = \{0000, 0001, 1110\}$. The set S has two different column types, represented by $(0, 0, 1)^T$, call it type 1, and $(0, 1, 0)^T$, call it type 2. There are three columns of type 1 and one column of type 2. The solution 0110 for S can be encoded by two variables $x[1] = 2$ and $x[2] = 0$.

We next introduce m variables $y \in \{0, 1, \dots, n\}^m$ that shall be equal to the Hamming distances of each input string s_i , $i \in [m]$, to the solution $s^*(x)$ selected by x . To achieve this, we need a formula specifying the Hamming distance between the two strings s_i and $s^*(x)$, and this formula needs to be linear in x . This can be achieved as follows; for the sake of simplicity, we let $s_i[j] = 1$ if the column of type j has one in the i^{th} row and $s_i[j] = 0$ if it has zero in the i^{th} row: $d(s_i, s^*(x)) = \sum_{j=1}^{n'} (s_i[j] \cdot (e(j) - x[j]) + (1 - s_i[j]) \cdot x[j]) = \sum_{j=1}^{n'} (e(j) \cdot s_i[j] + (1 - 2s_i[j]) \cdot x[j]) = w_i + \sum_{j=1}^{n'} x[j] \cdot (1 - 2s_i[j])$, where we define $w_i := \sum_{j=1}^{n'} e(j) \cdot s_i[j]$, which denotes the number of ones in string s_i .

We can now formulate an appropriate CnIP. The variables are $x \in \mathbb{R}^{n'}$, $y \in \mathbb{R}^m$, and a dummy variable $z \in \mathbb{Z}$. The bounds ℓ, u for the variables are defined such that

- (1) for each $j \in [n']$ it holds that $0 \leq x[j] \leq e(j)$,
- (2) for each $i \in [m]$ it holds that $0 \leq y[i] \leq n$, and
- (3) there is virtually no constraint on z , that is, $-n' \cdot n + mn \leq z \leq n' \cdot n + mn$.

The objective function is defined as $f(x, y, z) = \sum_{i=1}^m y[i]^p$ which is clearly separable convex over the domain specified by ℓ and u . Finally, the constraint system $Et = b$, where $t^T = (x^T y^T z)$ is defined such that the first m constraints are $\sum_{j=1}^{n'} (x[j] \cdot (1 - 2s_i[j])) - y[i] = -w_i$, for each $i \in [m]$, and the last constraint is $\sum_{j=1}^{n'} x[j] + \sum_{i=1}^m y[i] + z = 0$ (note that this constraint can always be fulfilled by setting z accordingly).

By the above reasoning, an instance of p -HDC is a yes-instance if and only if $\min\{f(x) \mid Et = b \wedge \ell \leq t \leq u \wedge t \in \mathbb{Z}^{n'+n+1}\}$ is at most k^p . Plugging in the running time of Proposition 11, and using a polynomial-time algorithm for the continuous relaxation of the CnIP above [10], we obtain the following.

► **Theorem 13.** p -NORM HAMMING CENTROID can be solved in $m^{O(m^2)} \cdot (n \cdot m)^{O(1)}$ time.

3.3 A Factor-2 Approximation

It is known that by taking an input string that minimizes the largest Hamming distance over all input strings, CLOSEST STRING can be approximated within factor 2. Indeed, using a similar idea, we show that the minimization version of our p -HDC problem can also be approximated within factor 2. More specifically, we show that an input string which has minimum p -norm to all other input strings is a 2-approximate solution.

► **Proposition 14** (*). *The minimization variant of p -HDC can be approximated within factor 2 in polynomial time.*

4 Conclusion and Outlook

We analyzed the complexity of p -NORM HAMMING CENTROID for all fixed rational values p between $p = 1$ and $p = \infty$. We believe that the running time bounds established in this paper, of essentially $2^{\Theta(k^{\frac{p}{p+1}})} \cdot (nm)^{O(1)}$, connect the extreme points $p = 1$ and $p = \infty$ in

a very satisfying way. We did not consider the non-norm case of $0 < p < 1$, as it does not fit our clustering motivation very well. But this non-convex case might be of independent interest, and may be the subject of future work. Furthermore, we have focused here on the case where p is a fixed constant. It would also be interesting to study the case where p is part of the input.

An interesting generalization of CLOSEST STRING is CLOSEST SUBSTRING in which we seek a string s^* of a certain specified length such that each of the input strings has a substring which is close to s^* (see, e.g., Ma and Sun [31]). It would be interesting to see how our results carry over to this and other similar variants. Finally, the fact that the simple 2-factor approximation for CLOSEST STRING carries over to p -HDC may imply that there are similar connections for approximation algorithms. This warrants further investigation into whether p -HDC admits a PTAS.

References

- 1 Georgios Amanatidis, Nathanaël Barrot, Jérôme Lang, Evangelos Markakis, and Bernard Ries. Multiple referenda and multiwinner elections using hamming distances: Complexity and manipulability. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '15)*, pages 715–723, 2015.
- 2 Yossi Azar, Leah Epstein, Yossi Richter, and Gerhard J. Woeginger. All-norm approximation algorithms. *Journal of Algorithms*, 52(2):120–133, 2004. doi:10.1016/j.jalgor.2004.02.003.
- 3 Gleb Beliakov, Humberto Bustince Sola, and Tomasa Calvo. *A Practical Guide to Averaging Functions*, volume 329 of *Studies in Fuzziness and Soft Computing*. Springer, 2016.
- 4 Paul S. Bradley, Olvi L. Mangasarian, and W. Nick Street. Clustering via concave minimization. In *Proceedings of Advances in Neural Information Processing Systems 9 (NIPS '96)*, pages 368–374, 1996.
- 5 Steven J. Brams, D. Marc Kilgour, and M. Remzi Sanver. A minimax procedure for negotiating multilateral treaties. In Rudolf Avenhaus and I. William Zartman, editors, *Diplomacy Games: Formal Models and International Negotiations*, pages 265–282. Springer, 2007. doi:10.1007/978-3-540-68304-9_14.
- 6 Margaret L. Brandeau and Samuel S. Chiu. Parametric facility location on a tree network with an L_p -norm cost function. *Transportation Science*, 22(1):59–69, 1988.
- 7 Jiehua Chen, Danny Hermelin, and Manuel Sorge. A note on clustering aggregation. Technical report, arXiv:1807.08949, 2018. arXiv:1807.08949.
- 8 Jiehua Chen, Danny Hermelin, and Manuel Sorge. On computing centroids according to the p -norms of hamming distance vectors. Technical report, arXiv:1807.06469, 2018. arXiv:1807.06469.
- 9 Zhi-Zhong Chen, Bin Ma, and Lusheng Wang. A three-string approach to the closest string problem. *Journal of Computer and System Sciences*, 78(1):164–178, 2012. doi:10.1016/j.jcss.2011.01.003.
- 10 Sergei Chubanov. A polynomial-time descent method for separable convex optimization problems with linear constraints. *SIAM Journal on Optimization*, 26(1):856–889, 2016. doi:10.1137/14098524X.
- 11 Gérard D. Cohen, Iiro S. Honkala, Simon Litsyn, and Antoine Lobstein. *Covering Codes*, volume 54. North-Holland, 1997.
- 12 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.
- 13 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Lower bounds for approximation schemes for closest string. In *Proceedings of the 15th Scandinavian Workshop on Algorithm Theory (SWAT '16)*, volume 53 of *LIPICs*, pages 12:1–12:10. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

- 14 Amit Deshpande, Madhur Tulsiani, and Nisheeth K. Vishnoi. Algorithms and hardness for subspace approximation. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '11)*, pages 482–496, 2011.
- 15 Martin Dörnfelder, Jiong Guo, Christian Komusiewicz, and Mathias Weller. On the parameterized complexity of consensus clustering. *Theoretical Computer Science*, 542:71–82, 2014. doi:10.1016/j.tcs.2014.05.002.
- 16 Piotr Faliszewski, Piotr Skowron, Arkadii Slinko, and Nimrod Talmon. Committee scoring rules: Axiomatic characterization and hierarchy. *ACM Transactions on Economics and Computation*, 7(1):3:1–3:39, 2019.
- 17 Piotr Faliszewski, Piotr Skowron, Arkadii Slinko, and Nimrod Talmon. Multiwinner rules on paths from k -Borda to Chamberlin-Courant. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI '17)*, pages 192–198. AAAI Press, 2017.
- 18 Moti Frances and Ami Litman. On covering problems of codes. *Theory of Computing Systems*, 30:113–119, 1997.
- 19 Michael R. Garey and David S. Johnson. *Computers and Intractability—A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
- 20 Dennis C. Ghiglia and Louis A. Romero. Minimum l_p -norm two-dimensional phase unwrapping. *Journal of the Optical Society of America A*, 13(10):1999–2013, 1996.
- 21 René Gonin and Arthur H. Money. *Nonlinear L_p -norm Estimation*. Marcel Dekker, Inc., 1989.
- 22 Jens Gramm, Rolf Niedermeier, and Peter Rossmanith. Fixed-parameter algorithms for Closest String and related problems. *Algorithmica*, 37(1):25–42, 2003.
- 23 Venkatesan Guruswami, Prasad Raghavendra, Rishi Saket, and Yi Wu. Bypassing UGC from some optimal geometric inapproximability results. *ACM Transactions on Algorithms*, 12(1):6:1–6:25, 2016.
- 24 Richard W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2), 1950.
- 25 Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.
- 26 D. Marc Kilgour. Approval balloting for multi-winner elections. In J.-F. Laslier and M.R. Sanver, editors, *Handbook on Approval Voting, Studies in Choice and Welfare*, chapter 6, pages 105–124. Springer, 2010.
- 27 Marius Kloft, Ulf Brefeld, Sören Sonnenburg, Pavel Laskov, Klaus-Robert Müller, and Alexander Zien. Efficient and accurate l_p -norm multiple kernel learning. In *Proceedings of Advances in Neural Information Processing Systems 22 (NIPS '09)*, pages 997–1005, 2009.
- 28 Dušan Knop, Martin Koutecký, and Matthias Mnich. Combinatorial n -fold integer programming and applications. Technical report, arXiv:1705.08657, 2017. arXiv:1705.08657.
- 29 J. Kevin Lanctôt, Ming Li, Bin Ma, Shaojiu Wang, and Louxin Zhang. Distinguishing string selection problems. *Information and Computation*, 185(1):41–55, 2003.
- 30 Robert F. Love, J. James G. Morris, and George O. Wesolowsky. *Facilities Location: Models & Methods*. North-Holland, 1988.
- 31 Bin Ma and Xiaoming Sun. More efficient algorithms for closest string and substrings problems. *SIAM Journal on Computing*, 39(4):1432–1443, 2009.
- 32 J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.
- 33 A. H. Money, J. F. Affleck-Graves, M. L. Hart, and G. D. I. Barr. The linear regression model: l_p norm estimation and the choice of p . *Journal of Communications in Statistics—Simulation and Computation*, 11(1):89–109, 1982.
- 34 Yurii Nesterov and Arkadii Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Series: Studies in Applied and Numerical Mathematics. Society for Industrial and Applied Mathematics, 1994.
- 35 Fanny Pascual, Krzysztof Rzdca, and Piotr Skowron. Collective schedules: Scheduling meets computational social choice. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 18)*, pages 667–675, 2018. URL: <http://dl.acm.org/citation.cfm?id=3237482>.

28:16 On Computing Centroids According to the p -Norms of Hamming Distance Vectors

- 36 Pavel A. Pevzner. *Computational Molecular Biology—An Algorithmic Approach*. MIT Press, 2000.
- 37 Ron M. Roth. *Introduction to Coding Theory*. Cambridge University Press, 2006.
- 38 Douglas R. Shier and Perino M. Dearing. Optimal locations for a class of nonlinear, single-facility location problems on a network. *Operations Research*, 31(2):292–303, 1983.
- 39 Shankar Sivarajan. A generalization of the minisum and minimax voting methods. *SIAM Undergraduate Research Online*, 11, 2018. doi:10.1137/16S014870.
- 40 Wen-Jun Zeng, Hing-Cheung So, and Abdelhak M. Zoubir. An ℓ_p -norm minimization approach to time delay estimation in impulsive noise. *Digital Signal Processing*, 23(4):1247–1254, 2013.

Non-Cooperative Rational Interactive Proofs

Jing Chen

Stony Brook University, Stony Brook, NY 11794-4400, USA
jingchen@cs.stonybrook.edu

Samuel McCauley

Williams College, Williamstown, MA 01267, USA
sam@cs.williams.edu

Shikha Singh

Williams College, Williamstown, MA 01267, USA
shikha@cs.williams.edu

Abstract

Interactive-proof games model the scenario where an honest party interacts with powerful but strategic provers, to elicit from them the correct answer to a computational question. Interactive proofs are increasingly used as a framework to design protocols for computation outsourcing.

Existing interactive-proof games largely fall into two categories: either as games of cooperation such as multi-prover interactive proofs and cooperative rational proofs, where the provers work together as a team; or as games of conflict such as refereed games, where the provers directly compete with each other in a zero-sum game. Neither of these extremes truly capture the strategic nature of service providers in outsourcing applications. How to design and analyze non-cooperative interactive proofs is an important open problem.

In this paper, we introduce a mechanism-design approach to define a multi-prover interactive-proof model in which the provers are *rational* and *non-cooperative* – they act to maximize their expected utility given others’ strategies. We define a strong notion of backwards induction as our solution concept to analyze the resulting extensive-form game with imperfect information.

We fully characterize the complexity of our proof system under different *utility gap* guarantees. (At a high level, a utility gap of u means that the protocol is robust against provers that may not care about a utility loss of $1/u$.) We show, for example, that the power of non-cooperative rational interactive proofs with a polynomial utility gap is exactly equal to the complexity class P^{NEXP} .

2012 ACM Subject Classification Theory of computation → Algorithmic game theory and mechanism design; Theory of computation → Interactive proof systems; Theory of computation → Computational complexity and cryptography

Keywords and phrases non-cooperative game theory, extensive-form games with imperfect information, refined sequential equilibrium, rational proofs, interactive proofs

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.29

Related Version A full version of the paper is available at <https://arxiv.org/abs/1708.00521>.

Funding This work has been partially supported by NSF CAREER Award CCF 1553385, CNS 1408695, CCF 1439084, IIS 1247726, IIS 1251137, CCF 1217708, by Sandia National Laboratories, by the European Research Council under the European Union’s 7th Framework Programme (FP7/2007-2013) / ERC grant agreement no. 614331, and a Zuckerman STEM Fellowship.

1 Introduction

Game theory has played a central role in analyzing the conflict and cooperation in interactive proof games. These games model the scenario where an honest party interacts with powerful but strategic agents, to elicit from them the correct answer to a computational question. The extensive study of these games over decades has fueled our understanding of important



© Jing Chen, Samuel McCauley, and Shikha Singh;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 29; pp. 29:1–29:16



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

complexity classes (e.g., [4, 16, 22–24, 26, 27, 37]). From a modern perspective, these games capture the essence of computation outsourcing – the honest party is a client outsourcing his computation to powerful rational service providers in exchange for money.

In this paper, we consider a natural type of interactive-proof game. For the moment, let us call our client Arthur. Arthur hires a service provider Merlin to solve a computational problem for him, and hires a second service provider Megan to cross-check Merlin’s answer. Arthur wants the game (and associated payments) to be designed such that if Merlin gives the correct answer, Megan agrees with him; however, if Merlin cheats and gives a wrong answer, Megan is incentivized to contradict him, informing Arthur of Merlin’s dishonesty. This means that Merlin and Megan are not purely cooperative nor purely competitive. Each is simply a rational agent who wants to maximize their own utility.

This is a mechanism design problem – how can Arthur incentivize non-cooperative rational agents (Merlin and Megan) to give truthful answers to his questions, helping him solve a computational problem? This problem is the focus of our paper.

Structure of the game

We borrow the structure and terminology of interactive proofs [3, 6, 29], as was done in previous work on rational proofs [1, 2, 11, 12, 17–19, 31, 32] and refereed games [16, 22, 24–26, 35, 40]. We call Arthur the **verifier** and assume that he is computationally bounded (he may be probabilistic, but must run in polynomial time). Arthur’s coin flips are treated as Nature moves in the game. We call Merlin and Megan the **provers**; they have unbounded computational power.

The verifier exchanges messages with the provers in order to determine the answer to a decision problem. The exchange proceeds in rounds: in a round, either a verifier sends a message to all provers or receives a response from each. The provers cannot observe the messages exchanged between the verifier and other provers.

At the end, the verifier gives a payment to *each* prover. Our goal is to design protocols and payments such that, under an appropriate solution concept of the resulting game, the provers’ best strategies lead the verifier to the correct answer.

The interactive protocols described above form an extensive-form game of imperfect information. To analyze them, we essentially use a strong notion of backward induction as our solution concept. We refine it further by eliminating strategies that are weakly dominated on “subgames” within the entire game. We define the solution concept formally in Section 2.1.

Comparison to previous work

The model of our games is based on interactive proof systems [3, 29], in which a verifier exchanges messages with untrustworthy provers and at the end either accepts or rejects their claim. Interactive proofs guarantee that, roughly speaking, the verifier accepts a truthful claim with probability at least $2/3$ (**completeness**) and no strategy of the provers can make the verifier accept a false claim with probability more than $1/3$ (**soundness**).

The study of interactive proofs has found extensive applications in both theory and practice. Classical results on IPs have led us to better understand complexity classes through characterizations such as $IP = PSPACE$ [37, 43] and $MIP = NEXP$ [4, 23, 27], and later led to the important area of probabilistically checkable proofs [44]. More recently, the study of IPs has resulted in extremely efficient (e.g., near linear or even logarithmic time) protocols for delegation of computation [7, 9, 15, 30, 41]. Such super-efficient IPs have brought theory closer to practice, resulting in “nearly practical” systems (e.g., see [8, 13, 45, 47]).

Indeed, interactive proofs are not only a fundamental theoretical concept but an indispensable framework to design efficient computation-outsourcing protocols.

Existing interactive-proof games

Interactive-proof systems with multiple provers have largely been studied as games that fall into two categories: either as games of cooperation such as MIP [6], cooperative multi-prover rational proofs (MRIP) [18], and variants [4, 10, 27, 30, 33], where the provers work together to convince the verifier of their joint claim; or as games of conflict such as refereed games [14–16, 22, 24, 26, 34], where the provers directly compete with each other to convince the verifier of their conflicting claims.

Both of these categories have limitations. In a game of cooperation, provers cannot be leveraged directly against each other. That is, the verifier cannot directly ask one prover if another prover is lying. On the other hand, in a game of conflict, such as refereed games, one prover must “win” the zero-sum game. Thus, such games need to assume that at least one prover – who must be the winning prover in a correct protocol – can be trusted to always tell the truth. Despite their limitations, both models have proved to be fundamental constructs to understand and characterize important complexity classes [4, 16, 18, 22, 26], and to design efficient computation outsourcing protocols [7, 8, 14, 15, 30].

1.1 Contributions and Results

In this paper, we introduce a new interactive-proof game, *non-cooperative rational interactive proofs* (*ncRIP*). This model generalizes multi-prover rational proofs [17–19].

Solution concept for ncRIP

We define a refinement of sequential equilibrium [36], **strong sequential equilibrium** (SSE), that essentially says that players’ beliefs about the histories that led them to an unreachable information set should be irrelevant to their best response. From a mechanism-design perspective, we want to design the protocols and payments that allow this strong guarantee to hold – letting the players’ best responses be unaffected by their beliefs.¹

Finally, we eliminate SSE strategies that are suboptimal within “subgames” by defining and enforcing a backward-induction-compatible notion of dominance. Roughly speaking, we say a protocol is a ncRIP if there exists a strategy profile of the provers that is a dominant SSE among the *subforms* of the extensive form game, and under this strategy the provers’ lead the verifier to the correct answer. We define the model formally in Section 2.

Utility gap for non-cooperative provers

Utility gap is a fundamental concept for rational proofs [2, 18, 19, 31] which is analogous to *soundness gap* in interactive proofs. It measures how robust a protocol is against the provers’ possible deviations from the desired strategy.

This notion is straightforward to define for cooperative rational protocols – they have a utility gap of u if the *total* expected payment decreases by $1/u$ whenever the provers report the wrong answer. In non-cooperative protocols, however, it is not a priori clear how to define such a payment loss or to choose which prover should incur the loss. A payment loss

¹ We believe that SSE is of independent interest as a solution concept for designing extensive-form mechanisms (e.g. [21, 28, 46]). In the full version of the paper, we prove important properties of SSE that may prove useful in future studies.

solely imposed on the total payment may not prevent some provers from deviating, and a loss solely imposed on the provers' final payments may not prevent them from deviating within subgames.

We define a meaningful notion of utility gap for ncRIP that is naturally incorporated in a backward-induction-compatible way to the dominant SSE concept.

Tight characterizations of ncRIP classes

In this paper, we completely characterize the power of non-cooperative rational proofs under different utility-gap guarantees.

We construct ncRIP protocols with constant, polynomial, and exponential utility gaps for powerful complexity classes, demonstrating the strength of our solution concept. Our protocols are simple and intuitive (requiring only a few careful tweaks from their cooperative counterparts), and are thus easy to explain and implement. However, proving their correctness involves analyzing the extensive-game (including subtleties in the incentives and beliefs of each player at each round) to show that the protocol meets the strong solution-concept and utility-gap requirements.

We then prove tight upper bounds for all three ncRIP classes. Proving tight upper bounds is the most technically challenging part of the paper. We prove the upper bounds by simulating the decisions of the verifier and provers with a Turing Machine. However, there are several obstacles to attain the correct bounds. For example, the polynomial randomness of the verifier can induce an exponential-sized game tree, which is too large to be verified by the polynomial-time machine in Theorems 1 and 2. Furthermore, an NEXP oracle cannot itself verify whether a strategy profile is a dominant SSE. The key lemma that helps us overcome these challenges is the pruning lemma (Lemma 13). At a high level, it shows that we can prune the nature moves of the verifier in the resulting game tree, while preserving the dominant-SSE and utility-gap guarantees.

Our results are summarized in Figure 1, where we use $O(1)$ -ncRIP, $\text{poly}(n)$ -ncRIP and $\text{exp}(n)$ -ncRIP to denote ncRIP classes with constant, polynomial and exponential utility gaps respectively. The notations are analogous for MRIP [17] (the cooperative variant). We characterize ncRIP classes via oracle Turing machines. In particular, $\mathbf{P}^{\text{NEXP}[O(1)]}$ is the class of languages decided by a polynomial-time Turing machine that makes $O(1)$ queries to an NEXP oracle, and $\text{EXP}^{\text{poly-NEXP}}$ is the class decided by an exponential-time Turing machine with polynomial-length queries to an NEXP oracle.

Note that lower and upper bounds for the case of exponential utility gap (that is, Theorem 3 and Corollary 6) are deferred to the full version of the paper.

▶ Theorem 1. $O(1)$ -ncRIP = $\mathbf{P}^{\text{NEXP}[O(1)]}$	▶ Corollary 4. $O(1)$ -ncRIP = $O(1)$ -MRIP
▶ Theorem 2. $\text{poly}(n)$ -ncRIP = \mathbf{P}^{NEXP}	▶ Corollary 5. $\text{poly}(n)$ -ncRIP \supseteq $\text{poly}(n)$ -MRIP
▶ Theorem 3. $\text{exp}(n)$ -ncRIP = $\text{EXP}^{\text{poly-NEXP}}$	▶ Corollary 6. $\text{exp}(n)$ -ncRIP = $\text{exp}(n)$ -MRIP

■ **Figure 1** Summary of our results.

Power of non-cooperative vs. cooperative and competitive provers

Interestingly, in the case of constant and exponential utility gap, the power of ncRIP and MRIP coincide. This can be explained by the power of adaptive versus non-adaptive queries in oracle Turing machines.

Indeed, our results reveal the main difference between non-cooperative and cooperative provers: the former can be used to handle adaptive oracle queries, the latter cannot (see [17, 18]). Intuitively, this makes sense – cooperative provers may collude across adaptive queries, answering some of them incorrectly to gain on future queries. On the other hand, non-cooperativeness allows us to treat the subgame involving the oracle queries as a separate game from the rest.

Our results also show that non-cooperative provers are more powerful than competing provers. Feige and Kilian [22] proved that the power of refereed games with imperfect information and perfect recall is equal to EXP.

2 Non-Cooperative Rational Interactive Proofs

In this section we introduce the model for ncRIP.

Notation

First, we review the structure of ncRIP protocols and related notation; this is largely the same as [18].

The decision problem being solved by an interactive proof is modeled as whether a given string x is in language L . An interactive protocol is a pair (V, \vec{P}) , where V is the **verifier**, $\vec{P} = (P_1, \dots, P_{p(n)})$ is the vector of $p(n)$ **provers**, where $p(n)$ is polynomial in $n = |x|$. The verifier runs in polynomial time and flips private coins. Each P_i is computationally unbounded. The verifier and provers are given the input x . Similar to classical multi-prover interactive proofs, the verifier can communicate with each prover privately, but no two provers can communicate with each other once the protocol begins.

In a **round**, either each prover sends a message to V , or V sends a message to each prover, and these two cases alternate. The length of each message $\ell(n)$, and the number of rounds $k(n)$ are both polynomial in n . The final transcript \vec{m} of the protocol is a random variable depending on r , the random string used by V . At the end of the communication, the verifier computes an **answer bit** $c \in \{0, 1\}$ for the membership of x in L based on x , r , and \vec{m} . V also computes a payment vector $\vec{R} = (R_1, R_2, \dots, R_{p(n)})$, where R_i is the payment given to P_i , $R_i \in [-1, 1]$, and the total $\sum_{i=1}^{p(n)} R_i \in [-1, 1]$ as well.² The protocol and the payment function \vec{R} are public knowledge.

Each prover P_i 's **strategy** at round j maps the transcript seen at the beginning of round j to the message he sends in that round. Let $s_i = (s_{i1}, \dots, s_{ik(n)})$ be the strategy of prover P_i , and $s = (s_1, \dots, s_{p(n)})$ be the **strategy profile** of the provers. Given input x , and strategy profile s , let $u_k(x, s, (V, \vec{P}))$ denote the expected payment of prover P_k in the protocol (V, \vec{P}) based on randomness r , input x and s ; if (V, \vec{P}) is clear from context, we shorten this to $u_k(x, s)$ or $u_k(s)$.

The protocol forms an extensive-form game with imperfect information and should be designed such that the provers are incentivized to reach an equilibrium that leads V to the correct answer bit c . We formalize the solution concept next.

² Negative payments are used to reflect punishment. The individual payments and the total payment can be shifted and scaled to lie in $[0, 1]$.

2.1 Solution concept for ncRIP

We want the solution concept for ncRIP to satisfy a strong notion of backward induction [39], a standard criterion applied to extensive-form games based on the common knowledge of rationality. Backwards induction refers to the condition of being “sequentially rational” in an extensive-form game, that is, each player must play his best response at each node where he has to move, even if his rationality implies that such a node will not be reached.

If an interactive protocol forms an extensive-form game of perfect information, it is easy to formalize this condition. A strategy s is **sequentially rational** or satisfies **backward induction**, if for every player i and every decision node of i , conditioned on reaching the decision node, s_i is a best response to s_{-i} , that is, $u_i(s_i, s_{-i}) \geq u_i(s'_i, s_{-i})$ for any strategy s'_i of prover i . In other words, s induces a best response at every subgame.³

In a game of imperfect information, the decision nodes corresponding to a player’s turn are partitioned into **information sets**, where the player is unable to distinguish between the possible histories within an information set. To reason about sequential rationality we need a probability distribution u_I on each information set I , so as to determine the players’ expected utility conditioned on reaching I and thus their best response at I . The probability distribution μ_I is referred to as the player’s **beliefs** about the potential histories leading to I .

Given a strategy profile s , beliefs u_I at **reachable information sets** (reached with non-zero probability under s) are derived from s using Bayes’ rule; this is a standard derivation used in most solution concepts for extensive-form games [39]. We sometimes write μ_I^s to emphasize that the beliefs depend on s .

Past work has introduced a variety of methods for defining the beliefs u_I^s at **unreachable information sets** I (i.e. information sets reached with probability zero under s); see e.g. [20, 36, 38, 42]. The most well-known is sequential equilibrium [36], which demands an explicit system of beliefs that satisfies a (somewhat artificial) consistency condition. Other equilibria, like trembling hand [42], reason implicitly about beliefs at unreachable information sets by assigning a negligible probability with which the player’s hand “trembles,” and reaches an otherwise-unreachable information set. Further refinements of these take the structure and payoffs of the game into account [5, 20, 38].

The treatment of beliefs at unreachable information sets in these solution concepts is often focused on ensuring that they can be used to analyze *every* extensive-form game. From a mechanism-design perspective, our focus is different – we want to design mechanisms in such a way that they admit much stronger equilibrium requirements, even if such an equilibrium cannot be used to analyze every game.

At a high-level, we want the players’ beliefs to be irrelevant in determining their best response at unreachable information sets. We call this notion **strong sequential rationality**. A strategy profile s is **strongly sequentially rational** if for every information set I , conditioned on reaching I , s_i is a best response to s_{-i} with respect to μ_I^s , where

- μ_I^s is derived using Bayes’s if I is reachable under s , and
- μ_I^s is *any* arbitrary probability distribution if I is unreachable under s .

In the full version of the paper, we show that this requirement is equivalent to saying that, at an unreachable information set I , s_i must be a best response to s_{-i} conditioned on reaching each history $h \in I$. In other words, at an unreachable information set I , each player must have a *single* action that is the best response to every possible history in I . We say a strategy profile is a **strong sequential equilibrium** (SSE) if it satisfies strong sequential rationality.

³ A subgame is a subtree that can be treated as a separate well-defined game. In a perfect-information game, every node starts a new subgame. “Backward induction” and “subgame-perfect equilibrium” are used interchangeably in the literature [28].

We refine our solution concept further to eliminate strategies that are weakly dominated within “subgames” of the entire game. This is crucial to deal with equilibrium selection, in particular, because the players’ cannot unilaterally deviate out of a suboptimal equilibria. We say an SSE s **weakly dominates** another SSE s' if, for any player i , $u_i(s) \geq u_i(s')$. A strategy s is **weakly dominant** if it dominates all SSEs. Next we eliminate SSEs that are weakly dominated in subgames of the entire game. We use the generalized notion of subgames, called **subforms**, defined by Kreps and Wilson [36] for extensive-form games with imperfect information.

To review the definition of subforms, we need further notation. Let H be the set of histories of the game. Recall that a history is a sequence (a^1, \dots, a^K) of actions taken by the players.⁴ For histories $h, h' \in H$, we say h has h' as a **prefix** if there exists some sequence of actions b^1, \dots, b^L (possibly empty) such that $h = (h', b^1, \dots, b^L)$. For a history $h \in H$, let $I(h)$ be the unique information set containing h .

For an information set I , let H_I be the set of all histories following I , that is, H_I is the set of all histories $h \in H$ such that h has a prefix in I . We say that H_I is a **subform rooted at I** if for every information set I' such that $I' \cap H_I \neq \emptyset$, it holds that $I' \subseteq H_I$. Roughly speaking, a subform H_I “completely contains” all histories of the information sets following I , so there is no information asymmetry between the players acting within H_I .

Thus, given a strategy profile, the subform H_I together with the probability distribution μ_I^s on I , can be treated as a well-defined game.

We say an SSE s **weakly dominates** SSE s' **on a subform H_I** if, for any player j acting in H_I , the expected utility of j under s_I in the game (H_I, μ_I^s) is greater than or equal to their utility under s'_I in the game $(H_I, \mu_I^{s'})$.

We eliminate weakly dominated strategies by imposing this dominance condition in a backward-induction-compatible way on the subforms as follows.

► **Definition 7** (Dominant Strong Sequential Equilibrium). *A strategy profile s is a dominant strong sequential equilibrium if s is an SSE and*

- *for every subform H_I of height 1: s weakly dominates s' on H_I for any SSE s'*
- *for every subform H_I subgame of height > 1 : s weakly dominates s' on H_I for any SSE s' that is a dominant SSE in all subforms of height at most $h - 1$.*

We are ready to define non-cooperative rational interactive proofs.

► **Definition 8** (Non-Cooperative Rational Interactive Proof). *Fix an arbitrary string x and language L . An interactive protocol (V, \vec{P}) is a non-cooperative rational interactive proof (ncRIP) protocol for L if there exists a strategy profile s of the provers that is a dominant SSE in the resulting extensive-form game, and under any dominant SSE, the answer bit c output by the verifier is correct (i.e., $c = 1$ iff $x \in L$) with probability 1, where the probability is taken over the verifier’s randomness.*

2.2 Utility Gap in ncRIP Protocols

In game theory, players are assumed to be perfectly rational and “sensitive” to arbitrarily small utility losses. In reality, some provers may not care about small losses. Such provers may not have sufficient incentive to reach a dominant SSE, and could end up leading the verifier to the wrong answer. To design ncRIP protocols that are robust against such “insensitive” provers, we define the notion of **utility gap**.

⁴ In the full version of the paper, we present a more formal treatment of the underlying extensive-form game, based on [39].

Informally, a utility gap of u means that if a strategy profile s leads the verifier to the wrong answer, there must exist a subform, such that some provers must lose at least a $1/u$ amount in their final individual payments (compared to their optimal strategy in that subform). As a consequence, these provers will not deviate to s , as long as they care about $1/u$ payment losses. We formalize this notion below. (We say a subform H_I is reachable under s if the information set I is reached under s with non-zero probability.)

► **Definition 9 (Utility Gap).** *Let (V, \vec{P}) be an ncRIP protocol for a language L and s^* be a dominant SSE of the resulting game. The protocol (V, \vec{P}) has an $\alpha(n)$ -utility gap or $\alpha(n)$ -gap, if for any strategy profile s' under which the answer bit c' is wrong, there exists a subform H_I reachable under s' , and a prover P_j acting in H_I who has deviated from s^* such that*

$$u_j(x, (s'_{-I}, s_I^*), (V, \vec{P})) - u_j(x, (s'_{-I}, s'_I), (V, \vec{P})) > 1/\alpha(n),$$

where s'_{-I} denotes the strategy profile s' outside subform H_I , that is, $s'_{-I} = s' \setminus s'_I$.

The class of languages that have an ncRIP protocol with **constant**, **polynomial** and **exponential** utility gap, are denoted by $O(1)$ -ncRIP, $\text{poly}(n)$ -ncRIP, and $\text{exp}(n)$ -ncRIP respectively.⁵ Note that $\alpha(n)$ gap corresponds to a payment loss of $1/\alpha(n)$, so an exponential utility gap is the weakest guarantee.

3 Lower Bounds: ncRIP Protocols with Utility Gap

In this section, we give an $O(1)$ -utility gap ncRIP protocol for the class NEXP and use it to give an $O(\alpha(n))$ -utility gap ncRIP protocol for the class $\text{P}^{\text{NEXP}[\alpha(n)]}$. Setting $\alpha(n)$ to be a constant or polynomial in n gives us $\text{P}^{\text{NEXP}[O(1)]} \subseteq O(1)\text{-ncRIP}$ and $\text{P}^{\text{NEXP}} \subseteq \text{poly}(n)\text{-ncRIP}$ respectively.

Here we argue correctness of our protocols at a high level; see the full version of the paper for formal proofs.

A constant-gap ncRIP protocol for NEXP

The ncRIP protocol for any language in NEXP is in Figure 2. The protocol uses the 2-prover 1-round MIP for NEXP [23] as a blackbox.⁶ The protocol in Figure 2 essentially forces the non-cooperative provers to coordinate by giving them identical payments. As a result, it is almost identical to the MRIP protocol for NEXP [18].

While the payment scheme is simple, in the analysis we have to open up the black-box MIP. In particular, if P_1 sends $c = 0$ in round 1, all the information sets of P_1 and P_2 in round 3 become unreachable. To show that an SSE exists, we show that the provers have a best response at these unreachable sets, which is argued based on the messages exchanged in the MIP protocol.

► **Lemma 10.** *Any language $L \in \text{NEXP}$ has a 2-prover 3-round $6/5$ -gap ncRIP protocol.*

⁵ These classes are formally defined by taking the union over languages with $\alpha(n)$ utility gap, for every $\alpha(n)$ that is constant, polynomial and exponential in n respectively.

⁶ It is also possible to give a scoring-rule based ncRIP protocol for NEXP, similar to MRIP [18]. However, such a protocol has an exponential utility gap.

For any input x and language $L \in \text{NEXP}$, the protocol (V, P_1, P_2) for L is:

1. P_1 sends a bit c to V . V outputs c at the end of the protocol.
2. If $c = 0$, then the protocol ends and the payments are $R_1 = R_2 = 1/2$.
3. Otherwise, V runs the classic 2-prover 1-round MIP protocol for NEXP [23] with P_1 and P_2 to prove if $x \in L$. If the MIP protocol accepts then $R_1 = 1, R_2 = 1$; else, $R_1 = -1, R_2 = -1$.

■ **Figure 2** A simple $O(1)$ -utility gap ncRIP protocol for NEXP .

An $O(\alpha(n))$ -gap ncRIP protocol for $\text{P}^{\text{NEXP}[\alpha(n)]}$

Using the above NEXP protocol as a subroutine, we give an ncRIP protocol with $O(\alpha(n))$ -utility gap for the class $\text{P}^{\text{NEXP}[\alpha(n)]}$. This protocol works for any function $\alpha(n)$ which (1) is a positive integer for all n , (2) is upper-bounded by a polynomial in n , and (3) is polynomial-time computable.⁷

► **Lemma 11.** *Any language $L \in \text{P}^{\text{NEXP}[\alpha(n)]}$ has a 3-prover 5-round ncRIP protocol that has a utility gap of $6/(5\alpha(n))$.*

The ncRIP protocol for any $L \in \text{P}^{\text{NEXP}[\alpha(n)]}$ is in Figure 3. It is fairly intuitive – V simulates the polynomial-time Turing machine directly, and uses the ncRIP protocol for NEXP for the oracle queries.

For any input x of length n , the protocol (V, \vec{P}) works as follows.

1. P_1 sends $(c, c_1, \dots, c_{\alpha(n)}) \in \{0, 1\}^{\alpha(n)+1}$ to V . V outputs c at the end of the protocol.
2. V simulates M on x using the bits $c_1, \dots, c_{\alpha(n)}$ as answers to NEXP queries $\phi_1, \dots, \phi_{\alpha(n)}$ generated by M respectively. If M accepts and $c = 0$ or M rejects and $c = 1$, then the protocol ends and $R_1 = -1, R_2 = R_3 = 0$.
3. V picks a random index i' from $\{1, \dots, \alpha(n)\}$ and sends $(i', \phi_{i'})$ to P_2 and P_3 .
4. V runs the 2-prover 3-round $O(1)$ -gap ncRIP protocol for NEXP (Figure 2) with P_2 and P_3 on $\phi_{i'}$. P_2 and P_3 get payments R_2 and R_3 based on the protocol. Let $c_{i'}^*$ be the answer bit in the NEXP protocol. If $c_{i'}^* \neq c_{i'}$, then $R_1 = 0$; otherwise $R_1 = 1$.

■ **Figure 3** An $O(\alpha(n))$ -utility gap ncRIP protocol for $\text{P}^{\text{NEXP}[\alpha(n)]}$.

We argue the correctness of this protocol at a high-level. Under any strategy of P_1 , the resulting NEXP queries in the protocol in Figure 3 are the roots of non-trivial subforms. Which of these subforms are reachable under a strategy profile s is determined solely by the strategy of P_1 . However, because weak dominance is imposed on all subforms in a bottom-up fashion, P_2 and P_3 must play their optimal strategy in these subforms regardless of their reachability – and therefore, they must play optimally for any strategy of P_1 . (This is one example of why ruling out weakly-dominated strategies in subforms in the definition of dominant SSEs is crucial to arguing correctness.) From the correctness of the NEXP protocol in Figure 2, we know that the optimal strategy of P_2 and P_3 is to compute the NEXP queries correctly. Given that the best response of P_2 and P_3 is to solve the NEXP queries correctly, and given that V randomly verifies 1 out of $\alpha(n)$ queries, P_1 must commit to correct answer bits in the first round, or risk losing a $O(1/\alpha(n))$ amount from his expected payment.

⁷ For Theorem 1 and Theorem 2, $\alpha(n)$ need only be a constant or polynomial in n . However, Lemma 11 holds for all $\alpha(n)$'s that are polynomial-time computable (given 1^n) and polynomially bounded, such as $\log n, \sqrt{n}$, etc.

If P_1 gives the correct answer bits in step 1, but P_2 or P_3 deviate within a subform corresponding to an NEXP query ϕ_q , then with probability $1/\alpha(n)$, V simulates the protocol in Figure 3 on ϕ_q , in which case they lose a constant amount of their expected payment.

4 Upper Bounds: ncRIP Protocols with Utility Gap

In this section, we prove matching upper bounds on the classes of ncRIP protocols with constant and polynomial utility gaps. In particular, we show that any language in $O(1)$ -ncRIP (or $\text{poly}(n)$ -ncRIP) can be decided by a polynomial-time Turing machine with a constant (resp. polynomial) number of queries to an NEXP oracle.

To simulate an ncRIP protocol, we need to find a strategy profile “close enough” to the dominant SSE so that the answer bit is still correct, i.e. a strategy profile that satisfies the utility-gap guarantee. We formalize this restatement of Definition 9 below.

► **Observation 12.** *Given input x and an ncRIP protocol (V, \vec{P}) with $\alpha(n)$ -utility gap, let s be a strategy profile such that for all reachable subforms H_I and all provers P_j acting in H_I ,*

$$u_j(x, r, (V, \vec{P}), (s_{-I}, s_I^*)) - u_j(x, r, (V, \vec{P}), (s_{-I}, s_I)) < \frac{1}{\alpha(n)},$$

where s^* is a dominant SSE. Then, the answer bit c under s must be correct.

There are several challenges involved in finding a strategy profile satisfying Observation 12.

First, the size of the game tree of any ncRIP protocol – small gap notwithstanding – can be exponential in n . Even if the polynomial-time machine considers a single strategy profile s at a time, since V can flip polynomially many coins, the part of the tree “in play” – the number of decision nodes reached with positive probability under s – can be exponential in n .

The second (and related) challenge is that of verifying whether a strategy profile is a dominant SSE. While the NEXP oracle can guess and verify an SSE, it cannot directly help with dominant SSEs. The polynomial-time machine must check using backward induction if an SSE is dominant on all its reachable subforms, which can again be exponential in n .

Finally, the polynomial-time machine needs to search through the exponentially large strategy-profile space in an efficient way to find one which leads to the correct answer.

In the remainder of the section we address these challenges. In Lemma 13 we show that we can prune the game tree, resolving the first two challenges. Then in Lemmas 17 and 18, we show how to efficiently search through the strategy-profile space.

Pruning Nature moves in ncRIP protocols

We now give our main technical lemma for the upper bound, which shows that we can limit ourselves to examining protocols with bounded game trees without loss of generality.

Recall that a verifier’s coin flips in an ncRIP protocol represent Nature moves in the resulting game. The problem is that a polynomial-time verifier can result in Nature moves that impose nonzero probabilities over exponentially many outcomes.

We prune the Nature moves of a verifier so that a polynomial-time Turing machine simulating an $\alpha(n)$ -utility-gap protocol can traverse the game tree reachable under a given s . This pruning operation takes exponential time (linear in the size of the game tree), and can be performed by the NEXP oracle.

► **Lemma 13 (Pruning Lemma).** *Let $L \in \alpha(n)$ -ncRIP and let (V, \vec{P}) be an ncRIP protocol for L with $\alpha(n)$ utility gap and $p(n)$ provers. Given an input x and a strategy s , the protocol (V, \vec{P}) can be transformed in exponential time to a new protocol (V', \vec{P}') , where*

- the probability distribution on the outcomes imposed by the Nature moves of V' for input x has $O(\alpha(n))$ support,
- if s is a dominant SSE of (V, \vec{P}) , then s induces a dominant SSE in (V', \vec{P}) ,
- $|u_j(x, s, (V, \vec{P})) - u_j(x, s, (V', \vec{P}))| < 1/(4\alpha(n))$ for all $j \in \{1, \dots, p(n)\}$, and
- the utility gap guarantee is preserved, that is, if the answer bit under s is wrong, then there exists a subform H_I in the game (V', \vec{P}) (reachable under s) and a prover P_j acting at H_I , such that P_j loses a $1/(2\alpha(n))$ amount in his expected payment compared to a strategy profile where s_I (induced by s on H_I) is replaced by s_I^* (the dominant SSE on H_I), keeping the strategy profile outside H_I , s_{-I} , fixed.

We prove Lemma 13 in several parts. First, given an input x and a strategy s of the provers, we show how to transform any verifier V that imposes a probability distribution over outcomes with exponential support into a verifier V' that imposes a probability distribution with $O(\alpha(n))$ support.

Let (V, \vec{P}) use $p(n)$ provers and let the running time of V be n^k for some constant k . There can be at most 2^{n^k} different payments that V can generate for a particular prover given input x . Given x and s , fix a prover index $j \in \{1, \dots, p(n)\}$. Let R_1, R_2, \dots, R_m be the payments generated by V on s for P_j . Let V 's randomness assign probability distribution $\mu = (p_1, p_2, \dots, p_m)$ to R_1, R_2, \dots, R_m respectively. Then, the expected payment of P_j under s is $u_j(x, s, (V, \vec{P})) = \sum_{i=1}^m p_i R_i$.

Recall that $u_j(x, s, (V, \vec{P})) \in [-1, 1]$ for all $1 \leq j \leq p(n)$. For each prover P_j , divide the interval $[-1, 1]$ into $4\alpha(n)$ intervals, each of length $1/(2\alpha(n))$. In other words, prover P_j 's i th interval is $[i/2\alpha(n), (i+1)/2\alpha(n)]$,⁸ for each $i \in \{-2\alpha(n), \dots, 2\alpha(n) - 1\}$.

We round the possible payments for P_j to a representative of the their corresponding interval. Specifically, we map each payment R_i to r_j as described in Equation 1. There

$$r_j = \begin{cases} \frac{4\ell+1}{4\alpha(n)} & \text{if } R_i \in \left[\frac{\ell}{2\alpha(n)}, \frac{2\ell+1}{4\alpha(n)} \right) \\ \frac{4\ell+3}{4\alpha(n)} & \text{if } R_i \in \left[\frac{2\ell+1}{4\alpha(n)}, \frac{\ell+1}{2\alpha(n)} \right) \end{cases} \quad (1) \quad p'_i = \begin{cases} \sum_{k \in T_j} p_k & \text{if } i = f(S(i)) \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

are potentially exponentially many different payments R_i , and only polynomially many different payments r_j , so several R_i must map to the same r_j . Let $T_j = \{i : R_i \text{ maps to } r_j\}$. Let $\mathcal{T} = \cup_j \{T_j\}$. Thus the total number of distinct r_j is $8\alpha(n)$, so $|\mathcal{T}| = O(\alpha(n))$. Let $S : \{1, \dots, m\} \rightarrow \mathcal{T}$ such that $S(i) = T_j$ if and only if $i \in T_j$.

For each $T_j \in \mathcal{T}$, let $f(T_j)$ denote a unique index in the set T_j . Without loss of generality, let $f(T_j)$ be the lowest index in T_j . We define a new probability distribution $\mu' = (p'_1, \dots, p'_h)$ over the payments R_1, \dots, R_h respectively, given by Equation 2. In particular, for every $T_j \in \mathcal{T}$, assign $R_{f(T_j)}$ probability $\sum_{k \in T_j} p_k$ and for every other index $\ell \in T_j$, $\ell \neq f(T_j)$, assign R_ℓ probability 0.

Define V' as a polynomial-time verifier that simulates all deterministic computation of V . For a fixed input x , V' imposes a probability distribution μ' with $O(\alpha(n))$ support for any probability distribution μ imposed by V . For other inputs, V' simulates V without any modification.

⁸ To include 1 as a possible payment, interval $2\alpha(n) - 1$ should be closed on both sides; we ignore this for simplicity.

29:12 Non-Cooperative Rational Interactive Proofs

Note that given input x , a strategy profile s and the protocol (V, \vec{P}) , transforming the distribution μ to μ' takes time linear in the size of the game tree, and thus exponential in n . (This means that an NEXP oracle, given x , can guess a particular s and perform the transformation.)

The remainder of the proof of Lemma 13 consists of the following three claims. We argue their correctness at a high-level and defer formal proofs to the full version.

First, we show that if the strategy profile s is a dominant SSE of (V, \vec{P}) , then s restricted to the pruned game tree of (V', \vec{P}) imposes a dominant SSE on (V', \vec{P}) as well.

► **Claim 14.** *Any dominant SSE s of the game formed by (V, \vec{P}) induces a dominant SSE in the game formed by (V', \vec{P}) .*

First, we prove that s is an SSE of (V', \vec{P}) . Suppose by contradiction that s is not a SSE of (V', \vec{P}) . Then there exists an information set I , such that, conditioned on reaching I , the prover acting at I can improve his expected payment by deviating (given his belief u'_I at I if I is reachable under s and for any belief he may hold at I if I is unreachable under s). Writing out their expected payments, accounting for the probabilistic transformation between V and V' , in both cases leads to a contradiction to the assumption that s was an SSE in (V, \vec{P}) . We then argue that a similar contradiction holds for proving that s is a dominant SSE of (V', \vec{P}) .

The following claim states that for a given s , the expected payments of the provers under (V, \vec{P}) and under (V', \vec{P}) are not too far off. This claim is one of the bullet points in Lemma 13, and will be used to prove Claim 16.

► **Claim 15.** *For all $j \in \{1, \dots, p(n)\}$, $|u_j(x, s, (V, \vec{P})) - u_j(x, s, (V', \vec{P}))| < 1/(4\alpha(n))$.*

With the above, we show that (V', \vec{P}) preserves utility gap guarantees.

► **Claim 16.** *Given input x , if the answer bit under s is wrong, then there exists a subform H_I reachable under s in (V', \vec{P}) and P_j acting at H_I , such that P_j 's expected payment under s is $\frac{1}{2\alpha(n)}$ less than his expected payment under (s_{-I}, s_I^*) , where s_I^* is a dominant SSE on H_I .*

Consider a strategy profile s^* that is a dominant SSE in the game tree of (V, \vec{P}) . Since s gives the wrong answer bit, from the $\alpha(n)$ -utility gap guarantee of (V, \vec{P}) and Definition 9, there exists a subform H_I reachable under s , such that a prover P_j acting in H_I loses $1/\alpha(n)$ in his expected payment under s compared to the strategy profile (s_{-I}, s_I^*) .

Using Claim 14, s^* also induces a dominant SSE in the game tree of (V', \vec{P}) . And since H_I is reachable under s in (V, \vec{P}) , it is reachable under s in (V', \vec{P}) as well. Finally, Claim 16 follows by applying Claim 15 twice: once to show that payments under V and V' are similar under s , and once to show that the payments are similar under (s_{-I}, s_I^*) . In the worst case this leads to a payment difference of $1/(4\alpha(n)) + 1/(4\alpha(n)) = 1/(2\alpha(n))$.

Using Lemma 13, given an $O(\alpha(n))$ -gap nCRIP protocol (where $\alpha(n)$ is constant or polynomial), a polynomial-time oracle Turing machine can use its NEXP oracle to guess a strategy profile s , prune the verifier's Nature moves, and report the resulting $O(\alpha(n))$ -support distribution bit-by-bit. Thus, it can simulate the new distribution and find the decision nodes that are reachable under s .

Searching through the strategy-profile space efficiently

The next question is: how should the polynomial-time Turing machine navigate the potential strategy-profile space (in polynomial time) to find the strategy profile that satisfies Observation 12? To cut down on the search space, we invoke a recurring idea: divide each prover's expected payment interval $[-1, 1]$, evenly into $8\alpha(n)$ **subintervals** of length $1/(4\alpha(n))$, and consider **subinterval profiles** (a tuple of subintervals, one for each prover).

► **Lemma 17.** *Given an input x and an ncRIP protocol (V, \vec{P}) with $\alpha(n)$ -utility gap, consider a subinterval profile $(L_1, \dots, L_{p(n)})$, where each $L_i = [k/(4\alpha), (k+1)/(4\alpha+1))$ denotes a subinterval of prover P_i in $[-1, 1]$, for some $k \in \{-2\alpha(n), \dots, 2\alpha(n) - 1\}$. Let s be an SSE that has an expected payment profile $\tilde{u}(x, s)$ such that $u_i(x, s) \in L_i$ for all $1 \leq i \leq p(n)$, and s does not satisfy Observation 12. Then the expected payment profile $\tilde{u}(x, s^*)$ under a dominant SSE s^* cannot lie in the same subinterval profile, that is, there exists a prover index j such that $u_j(x, s^*) \notin L_j$.*

Using Lemma 17, if the polynomial-time Turing machine is able to test *any* SSE s with $\tilde{u}(x, s)$ in a subinterval profile, for all subinterval profiles, then it is guaranteed to find one that satisfies Observation 12. This is because a dominant SSE of an ncRIP protocol is guaranteed to exist and its expected payment profile must belong to some subinterval profile.

However, there are still $O(\alpha(n))$ subintervals for each prover, and thus $O(\alpha(n)^{p(n)})$ total subinterval profiles. A polynomial-time machine cannot test SSEs for each of them.

To reduce the search space further, we show that it is sufficient to consider subintervals of the total expected payment rather than individual and test an SSE s for each of them. Recall that a SSE s is weakly dominant if for any player i and SSE s' , $u_i(s) \geq u_i(s')$.

► **Lemma 18.** *If a weakly-dominant SSE exists, then a strategy profile s is a weakly-dominant SSE if and only if s is an SSE and s maximizes the sum of utilities of all players among all SSEs.*

We are now ready to prove the upper bound for ncRIP classes with constant and polynomial utility gap. We defer formal details of the proof to the full version of the paper.

Constant utility gap

Using Lemma 13 and Lemma 18, simulating a constant-gap protocol using a $\text{P}^{\text{NEXP}[O(1)]}$ machine M is straightforward. We give a high-level overview below.

There are at most $O(1)$ subforms that are reachable under any strategy profile s , and the total expected payment of the provers conditioned on reaching these subforms will be in one of the $O(1)$ subintervals. Thus, there are $O(1)$ combinations of total expected payments on all subforms (including the whole game). M queries its NEXP oracle whether there exists an SSE that achieves that combination of total expected payments on those subforms, for all combinations. Then, M finds the maximum among all of the combinations that got a “yes.” Such a maximum is guaranteed to exist for an ncRIP protocol. Finally, M queries the oracle for the answer bit of the corresponding SSE by giving the dominant profile of total expected payments over the subgames.

► **Lemma 19.** $O(1)\text{-ncRIP} \subseteq \text{P}^{\text{NEXP}[O(1)]}$.

Polynomial utility gap

To simulate a polynomial-utility gap ncRIP protocol (V, \vec{P}) , using a P^{NEXP} machine M , we put to use all the structure we have established in this section.

For each of the $O(\alpha(n))$ total payment subintervals of the interval $[-1, 1]$ that correspond to an SSE, M does a recursive search to find an exact total expected payment $u(x, s)$ that is generated by an SSE. (We can restrict ourselves to $O(\alpha(n))$ oracle queries due to Lemma 18.) In particular, M queries the NEXP oracle: *Does there exist an SSE with total expected payment in the first half of the i th interval?* If the answer is *yes* then M recurses on the first half of the i th interval; M does not need to search the second half by Lemma 17. Otherwise

(if the answer is *no*) then M recurses on the second half. Thus, in polynomial time and with polynomial queries, M can find an exact $u(x, s)$ for an SSE s in the subinterval using the power of its *adaptive* queries.

Next, M simulates the protocol (V, \vec{P}) with the help of the oracle, under the SSE s for a given subinterval. Lemma 13 is crucial for M to simulate the verifier’s moves, because V in general can induce exponential-size distributions. M traverses the tree reachable under s “top-down” using the oracle to learn the pruned distributions and provers’ moves. Finally, M goes “bottom-up” to test whether s satisfies Observation 12 on all its reachable subgames.

► **Lemma 20.** $\text{poly}(n)\text{-ncRIP} \subseteq \text{P}^{\text{NEXP}}$.

References

- 1 Pablo Daniel Azar and Silvio Micali. Rational proofs. In *Proceedings of the Forty-Fourth Annual Symposium on Theory of Computing (STOC)*, pages 1017–1028, 2012.
- 2 Pablo Daniel Azar and Silvio Micali. Super-efficient rational proofs. In *Proceedings of the Fourteenth Annual ACM conference on Electronic Commerce (EC)*, pages 29–30, 2013.
- 3 László Babai. Trading group theory for randomness. In *Proceedings of the Seventeenth annual ACM symposium on Theory of Computing (STOC)*, pages 421–429, 1985.
- 4 László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational complexity*, 1(1):3–40, 1991.
- 5 Jeffrey S Banks and Joel Sobel. Equilibrium selection in signaling games. *Econometrica: Journal of the Econometric Society*, pages 647–661, 1987.
- 6 Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC)*, pages 113–131, 1988.
- 7 Nir Bitansky and Alessandro Chiesa. Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In *Advances in Cryptology–CRYPTO 2012*, pages 255–272. Springer, 2012.
- 8 Andrew J Blumberg, Justin Thaler, Victor Vu, and Michael Walfish. Verifiable computation using multiple provers. *IACR Cryptology ePrint Archive*, 2014:846, 2014.
- 9 Benjamin Braun, Ariel J Feldman, Zuocheng Ren, Srinath Setty, Andrew J Blumberg, and Michael Walfish. Verifying computations with state. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 341–357, 2013.
- 10 Jin-yi Cai, Anne Condon, and Richard J Lipton. On games of incomplete information. *Theoretical Computer Science*, 103(1):25–38, 1992.
- 11 Matteo Campanelli and Rosario Gennaro. Sequentially Composable Rational Proofs. In *International Conference on Decision and Game Theory for Security*, pages 270–288, 2015.
- 12 Matteo Campanelli and Rosario Gennaro. Efficient Rational Proofs for Space Bounded Computations. In *International Conference on Decision and Game Theory for Security*, pages 53–73, 2017.
- 13 Ran Canetti, Ben Riva, and Guy N Rothblum. Practical delegation of computation using multiple servers. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 445–454, 2011.
- 14 Ran Canetti, Ben Riva, and Guy N Rothblum. Two 1-round protocols for delegation of computation. In *International Conference on Information Theoretic Security*, pages 37–61, 2012.
- 15 Ran Canetti, Ben Riva, and Guy N Rothblum. Refereed delegation of computation. *Information and Computation*, 226:16–36, 2013.
- 16 Ashok K Chandra and Larry J Stockmeyer. Alternation. In *17th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 98–108, 1976.
- 17 Jing Chen, Samuel McCauley, and Shikha Singh. Rational Proofs with Multiple Provers (Full Version). *arXiv preprint arXiv:1504.08361*, 2015. [arXiv:1504.08361](https://arxiv.org/abs/1504.08361).

- 18 Jing Chen, Samuel McCauley, and Shikha Singh. Rational Proofs with Multiple Provers. In *Proceedings of the Seventh Innovations in Theoretical Computer Science Conference (ITCS)*, pages 237–248, 2016.
- 19 Jing Chen, Samuel McCauley, and Shikha Singh. Efficient Rational Proofs with Strong Utility-Gap Guarantees. In *International Symposium on Algorithmic Game Theory (SAGT)*, pages 150–162. Springer, 2018.
- 20 In-Koo Cho and David M Kreps. Signaling games and stable equilibria. *The Quarterly Journal of Economics*, 102(2):179–221, 1987.
- 21 John Duggan. An extensive form solution to the adverse selection problem in principal/multi-agent environments. *Review of Economic Design*, 3(2):167–191, 1998.
- 22 Uriel Feige and Joe Kilian. Making games short. In *Proceedings of the Twenty-Ninth Annual ACM Symposium On Theory of Computing (STOC)*, pages 506–516, 1997.
- 23 Uriel Feige and László Lovász. Two-prover one-round proof systems: their power and their problems. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing (STOC)*, pages 733–744, 1992.
- 24 Uriel Feige and Adi Shamir. Multi-oracle interactive protocols with constant space verifiers. *Journal of Computer and System Sciences*, 44(2):259–271, 1992.
- 25 Uriel Feige, Adi Shamir, and Moshe Tennenholtz. The noisy oracle problem. In *Proceedings of the Tenth Annual Conference on Advances in Cryptology (CRYPTO)*, pages 284–296, 1990.
- 26 Joan Feigenbaum, Daphne Koller, and Peter Shor. A game-theoretic classification of interactive complexity classes. In *Proceedings of Tenth Annual IEEE Structure in Complexity Theory Conference*, pages 227–237, 1995.
- 27 Lance Fortnow, John Rompel, and Michael Sipser. On the power of multi-prover interactive protocols. *Theoretical Computer Science*, 134(2):545–557, 1994.
- 28 Jacob Glazer and Motty Perry. Virtual implementation in backwards induction. *Games and Economic Behavior*, 15(1):27–32, 1996.
- 29 S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.*, 18(1), 1989.
- 30 Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing (STOC)*, pages 113–122, 2008.
- 31 Siyao Guo, Pavel Hubáček, Alon Rosen, and Margarita Vald. Rational arguments: single round delegation with sublinear verification. In *Proceedings of the Fifth Annual Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 523–540, 2014.
- 32 Siyao Guo, Pavel Hubáček, Alon Rosen, and Margarita Vald. Rational sumchecks. In *Theory of Cryptography Conference*, pages 319–351, 2016.
- 33 Tsuyoshi Ito and Thomas Vidick. A multi-prover interactive proof for NEXP sound against entangled provers. In *53rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 243–252, 2012.
- 34 Gillat Kol and Ran Raz. Competing provers protocols for circuit evaluation. In *Proceedings of the Fourth Annual Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 473–484, 2013.
- 35 Daphne Koller and Nimrod Megiddo. The complexity of two-person zero-sum games in extensive form. *Games and economic behavior*, 4(4):528–552, 1992.
- 36 David M Kreps and Robert Wilson. Sequential equilibria. *Econometrica: Journal of the Econometric Society*, pages 863–894, 1982.
- 37 Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)*, 39(4):859–868, 1992.
- 38 Andrew McLennan. Justifiable beliefs in sequential equilibrium. *Econometrica: Journal of the Econometric Society*, pages 889–904, 1985.
- 39 Martin J Osborne and Ariel Rubinstein. *A course in game theory*. MIT press, 1994.

29:16 Non-Cooperative Rational Interactive Proofs

- 40 John H Reif. The complexity of two-player games of incomplete information. *Journal of Computer and System Sciences*, 29(2):274–301, 1984.
- 41 Guy N Rothblum, Salil Vadhan, and Avi Wigderson. Interactive proofs of proximity: delegating computation in sublinear time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 793–802, 2013.
- 42 Reinhard Selten. Reexamination of the perfectness concept for equilibrium points in extensive games. *International journal of game theory*, 4(1):25–55, 1975.
- 43 Adi Shamir. $IP = PSPACE$. *J. ACM*, 39(4):869–877, 1992.
- 44 Madhu Sudan. Probabilistically checkable proofs. *Communications of the ACM*, 52(3):76–84, 2009.
- 45 Justin Thaler, Mike Roberts, Michael Mitzenmacher, and Hanspeter Pfister. Verifiable Computation with Massively Parallel Interactive Proofs. In *HotCloud*, 2012.
- 46 Hannu Vartiainen. Subgame perfect implementation of voting rules via randomized mechanisms. *Social Choice and Welfare*, 29(3):353–367, 2007.
- 47 Michael Walfish and Andrew J Blumberg. Verifying computations without reexecuting them. *Communications of the ACM*, 58(2):74–84, 2015.

Stronger ILPs for the Graph Genus Problem

Markus Chimani 

Theoretical Computer Science, Osnabrück University, Germany
markus.chimani@uos.de

Tilo Wiedera 

Theoretical Computer Science, Osnabrück University, Germany
tilo.wiedera@uos.de

Abstract

The minimum genus of a graph is an important question in graph theory and a key ingredient in several graph algorithms. However, its computation is NP-hard and turns out to be hard even in practice. Only recently, the first non-trivial approach – based on SAT and ILP (integer linear programming) models – has been presented, but it is unable to successfully tackle graphs of genus larger than 1 in practice.

Herein, we show how to improve the ILP formulation. The crucial ingredients are two-fold. First, we show that instead of modeling rotation schemes explicitly, it suffices to optimize over partitions of the (bidirected) arc set A of the graph. Second, we exploit the cycle structure of the graph, explicitly mapping short closed walks on A to faces in the embedding.

Besides the theoretical advantages of our models, we show their practical strength by a thorough experimental evaluation. Contrary to the previous approach, we are able to quickly solve many instances of genus > 1 .

2012 ACM Subject Classification Mathematics of computing → Graphs and surfaces; Mathematics of computing → Graph algorithms; Theory of computation → Linear programming

Keywords and phrases algorithm engineering, genus, integer linear programming

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.30

Funding Supported by the German Research Foundation (DFG) grant CH 897/2-2.

1 Introduction

The (orientable) genus of a graph G is the smallest number γ such that G can be embedded on an orientable surface of genus γ . The problem of determining γ is the *graph genus problem*.

Algorithmically exploiting a low graph genus is a vivid research field that spawned many results. It plays a key role for the complexity of certain problems, in particular w.r.t. polynomial-time approximation schemes (PTAS) and fixed parameter tractability (FPT) [8]. Algorithms tailored to achieve faster runtime on planar graphs can often be extended for the bounded genus setting (given a corresponding embedding, see below). For example, a bounded graph genus leads to: linear-time graph isomorphism testing [7]; FPT runtime for dominating set [15]; subexponential FPT runtime for many bidimensional problems, including vertex cover and variants of dominating set [13]; a quasi-PTAS for capacitated vehicle routing [1]; stronger preprocessing for several Steiner problems [29]; and many more.

Apart from such algorithmic uses, the problem is of independent interest in graph theory, where one is concerned with finding the genus of certain graph families (or even single graphs) [3–5, 12, 17, 18, 21, 22, 24, 26, 30–33, 37]. Typically, this involves induction and tedious arguments for the base cases.

Although genus is one of the best-established measures for non-planarity, its NP-hardness was proved relatively late in 1989 by Thomassen [35]. In contrast to other non-planarity measures like crossing number or skewness (or equivalently, maximum planar subgraph), and even compared to the *maximum* graph genus problem, there has been little algorithmic



© Markus Chimani and Tilo Wiedera;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 30; pp. 30:1–30:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

progress. In 1988, Thomassen gave an algorithm that computes the genus of a restricted class of graphs in polynomial time [36] (graphs that have so-called LEW-embeddings). The problem is in FPT w.r.t. its natural parameterization due to Robertson and Seymour [34]; Mohar gave a constructive proof for such an FPT-algorithm [25]. Still, the proof spans seven papers of more than 100 pages in total. It was later simplified and extended to graphs with bounded treewidth [19]. It remains open to find a practically feasible FPT-algorithm [27]. No general approximation algorithm is known, although some progress was recently made for restricted variants and closely related problems [6, 20, 23]. The first general exact algorithm – beyond explicit enumeration – was described only recently [2]. It uses ILP/SAT solving to optimize over all rotation systems of the input, but fails to compute genera higher than 1 in practice. Overall, there is no practical, non-trivial algorithm to find low-genus embeddings, not even heuristically. Clearly, this is a main stumbling block for applying any of the genus-based algorithms mentioned above in practice.

Contribution. In this paper, we use the existing ILP as a starting point to develop a practically feasible model. To this end, we establish two novel core ideas. The original approach needs to model the embedding’s rotation scheme (see below) explicitly, essentially considering a Hamiltonian cycle problem for each graph node. We show how to only consider partitions of the bidirected arc set to deduce a rotation scheme; among other benefits, this allows us to use fewer variables. The second concept is to explicitly consider the cycle structure of the graph, by examining short, closed walks of the graph. Overall, we obtain both theoretically and practically stronger LP-relaxations. We thus present the first approach to compute the genus of (reasonably sized) graphs in practice, even for genera > 1 .

2 Preliminaries

We only need to consider biconnected (since genus is additive over biconnected components), simple graphs where each node has degree ≥ 3 : Given an undirected graph, we obtain its *bidirected* counterpart by creating two oppositely directed arcs uv, vu for each edge $\{u, v\}$. Given a graph H , we denote its nodes, edges, and arcs by $V(H)$, $E(H)$, and $A(H)$, respectively. For the input graph G , we may simply write V , E , and A . A *closed walk* c on a bidirected graph G is a set of arcs such that the induced subgraph $G[c]$ is connected and for each node the number of entering arcs is equal to that of leaving arcs. A *cycle* is a closed walk that visits no node more than once. For a node v , we refer to the arcs entering (leaving) v as $\delta_+(v)$ ($\delta_-(v)$, respectively). Let $N(v)$ be the nodes adjacent to node v . Given two subsets W, U of nodes, we define $W \times_A U := (W \times U) \cap A$ as the arcs from W to U . Given a partition, we refer to its partition sets as *cells*; this term should not be confused with faces of an embedding. For $k \in \mathbb{N}$, let $[k] := \{1, \dots, k\}$.

Graph Embeddings. A *drawing* \mathcal{D} of an undirected graph $G = (V, E)$ on an orientable surface S is a set of points \mathcal{P} and curves \mathcal{C} on S , such that there are bijections $\mathcal{D}_V: V \rightarrow \mathcal{P}$ and $\mathcal{D}_E: E \rightarrow \mathcal{C}$. We require that for each edge uv the two endpoints of $\mathcal{D}_E(uv)$ are $\mathcal{D}_V(u), \mathcal{D}_V(v)$. A drawing is *crossing free* if for any two edges e, f , $\mathcal{D}_E(e)$ is disjoint from $\mathcal{D}_E(f)$ except for common endpoints. We say that a graph is *planar* if it admits a crossing free drawing on the sphere. A crossing free drawing \mathcal{D} induces a cyclic rotation scheme Π of edges around each node, an *embedding*. From a combinatorial perspective, Π fully specifies \mathcal{D} and the genus minimal surface that \mathcal{D} can be drawn on. The regions bounded by edges are the *faces* of Π .

Given an embedding Π , we may identify its faces by tracing them as follows: Consider the bidirected counterpart of G . Starting at a node v , we traverse an arc vw and continue with the cyclically succeeding arc leaving w (w.r.t. the order of the undirected edges in Π). We iterate this until we again arrive at vw , closing the traced face's boundary. Repeating this operation until all edges are traversed exactly once in both directions gives all faces of Π . The traversed arcs of a face form a closed walk. If a node (or edge) appears more than once on the walk, we call this node (or edge) *singular*. The face tracing will allow us to count the number f_Π of faces. Using the Euler characteristic $|V| + f_\Pi - |E| = 2 - 2\gamma$, we are able to determine the lowest-genus (i.e., γ) surface that Π can be drawn on. Algorithmically, we thus ask for an embedding yielding the maximum number of faces.

Integer Linear Programming. A *linear program* (LP) consists of a cost vector $c \in \mathbb{Q}^d$ together with a set of linear inequalities, called *constraints*, that define a polyhedron P in \mathbb{R}^d . In polynomial time, we can find a point $x \in P$ that maximizes the *objective value* $c^\top x$. Unless $P = NP$, this is no longer true when restricting x to have integral components; the so-modified problem is an *Integer Linear Program* (ILP). Conversely, the *LP-relaxation* of an ILP is obtained by dropping the integrality constraints on its variables. Typically, there are several ways to reduce a given NP-hard problem to an ILP. These reductions are referred to as *models*. To achieve good practical performance, one aims for a small model where the objective value of the LP-relaxation is close to that of the ILP. This is crucial, as ILP solvers rely on iteratively computing LP-relaxations to obtain dual bounds on the integral objective. When a model has too many constraints to be solved in its entirety, it is often sufficient to use only a reasonably sized constraint subset to achieve a provably optimal solution. Hence, we may dynamically add constraints, during the solving process; this is called *separation*. We say that model A is *at least as strong* as model B , if for all instances, the LP-relaxation's value of model A is no worse than that of B . If there also exists an instance for which A 's LP-relaxation yields a tighter value than that of B , we call A *stronger* than B .

Common Foundation and Predecessor Model. As the known model [2], ours will simulate the face tracing algorithm. As such, both models share a common foundation that we borrow from [2]. For the sake of completeness, we repeat its definition below: We use variables x_i that are 1 if and only if face i exists, and variables x_i^a that are 1 if and only if arc a participates in the boundary of face i . Let \bar{f} be an upper bound on the number of faces. We use the shorthands $x(I, A) := \sum_{i \in I, a \in A} x_i^a$ and $x(A) := x([\bar{f}], A)$; thereby, we may omit curly braces when providing sets of cardinality one. Consider the following (by itself insufficient!) model (1a–1e) that we call ILP_{Base} .

$$\max \quad \sum_{i=1}^{\bar{f}} x_i \tag{1a}$$

$$\text{s.t.} \quad 3x_i \leq x(i, A) \quad \forall i \in [\bar{f}] \tag{1b}$$

$$x(a) = 1 \quad \forall a \in A \tag{1c}$$

$$x(i, \delta_-(v)) = x(i, \delta_+(v)) \quad \forall i \in [\bar{f}], v \in V \tag{1d}$$

$$x_i, x_i^a \in \{0, 1\} \quad \forall i \in [\bar{f}], a \in A \tag{1e}$$

Following [2], ILP_{Base} ensures that the faces form a partition of the arc set such that each cell consists of at least three arcs and is a collection of closed walks.

It remains to ensure that the faces are consistent with some rotation scheme of the edges around the nodes. In [2], this is achieved via *predecessor* variables, and we denote the model by ILP_{Pre} in the following. It uses ILP_{Base} and additionally (2a–2d). The idea is to establish

30:4 Stronger ILPs for the Graph Genus Problem

a cyclic order of the incident edges of each node by a cut-based sub-ILP known from the traveling salesman problem.

$$x_i^{vw} \geq x_i^{uv} + p_{u,w}^v - 1, \quad x_i^{uv} \geq x_i^{vw} + p_{u,w}^v - 1 \quad \forall i \in [\bar{f}], v \in V, u, w \in N(v) : u \neq w \quad (2a)$$

$$\sum_{\substack{u \in N(v) \\ u \neq w}} p_{w,u}^v = 1, \quad \sum_{\substack{u \in N(v) \\ u \neq w}} p_{u,w}^v = 1 \quad \forall v \in V, w \in N(v) \quad (2b)$$

$$\sum_{u \in W, w \in N(v) \setminus W} p_{u,w}^v \geq 1 \quad \forall v \in V, W \subsetneq N(v) : \emptyset \neq W \quad (2c)$$

$$p_{u,w}^v \in \{0, 1\} \quad \forall v \in V, u, w \in N(v) : u \neq w \quad (2d)$$

3 Realizability Model

In contrast to the explicit modeling of an embedding in ILP_{Pre} , we establish the existence of an embedding *implicitly*. Our *realizability constraints* (see (3) later) require only the variables of ILP_{Base} . We first need some auxiliary concepts. A graph $G = (V, A)$ is *loopy* if it is directed, connected, each node has at least one incoming arc, and for each arc $uv \in A$ it holds that $\mathcal{K}(uv) := G[\{s : sv \in A\} \cup \{t : ut \in A\}]$ is a complete bipartite graph $K_{k,k}$ for some $k \in \mathbb{N}$, such that each arc is directed from the cell of u to that of v w.r.t. the bipartition.

► **Lemma 1.** *Loopy graphs are Hamiltonian, i.e., they contain a cycle traversing all nodes.*

Proof. We first show that any loopy graph allows a cycle cover of pairwise node-disjoint cycles. Assume it does not, consider a collection \mathcal{C} of pairwise node-disjoint cycles covering as many nodes as possible, and let v be an uncovered node. By loopiness, there exists a bipartition that induces two cells, an arc uv , and $\mathcal{K}(uv)$ has a node w (possibly $u = w$) in the cell of u , such that w is not contained in any cycle of \mathcal{C} : for any ℓ nodes of one cell in a cycle $c \in \mathcal{C}$, c also contains ℓ nodes of the other cell. As there are only finitely many nodes, we find a new cycle by iterating our argument, i.e., traversing the cycle's arcs in reverse order, thus increasing our cycle cover; a contradiction.

Now, let \mathcal{C} be a node-disjoint cycle cover. For a cycle $c \in \mathcal{C}$, an arc a connecting $V(c)$ with $V \setminus V(c)$ exists by connectivity. Hence, $\mathcal{K}(a)$ contains an arc uv of c and another arc wx of a different cycle $c' \in \mathcal{C}$. We join c with c' to a single cycle by replacing uv, wx with ux, wv . Iterating this yields the claim. ◀

► **Theorem 2.** *A graph G allows an embedding Π with at least ξ faces if and only if there exists a partition P of $A(G)$, such that*

- (a) *P consists of at least ξ cells;*
- (b) *every cell of P is a set of pairwise node-disjoint closed walks; and*
- (c) *for all subsets $X \subseteq P$, nodes $v \in V(G)$, and non-empty subsets $W \subsetneq N(v)$, we have $\{wv, vw : w \in W\} \neq \bigcup_{x \in X} \{a \in x : v \text{ incident to } a\}$.*

Before giving the formal proof, let us provide some intuition on property (c): It models that the rotation around each node v is consistent. While in ILP_{Pre} constraints (2a–2d) model the rotation explicitly, property (c) ensures the *existence* of a feasible rotation by preventing subcycles. In the rotation around v , any two subsequent arcs must share an edge or a face. Hence, there cannot exist a *proper* subset W of v 's neighbors, such that *exactly* the arcs between v and W belong to a subset X of faces. As shown below this is also sufficient.

Proof (of Theorem 2). (\implies) We obtain P by creating a cell for each face f of Π : it contains exactly the arcs traversed by f . This satisfies (a) and (b). Assume that (c) is not satisfied, i.e., there exist X, v, W (following the above selection rules) such that

$\{wv, vw : w \in W\} = \bigcup_{x \in X} \{a \in x : v \text{ incident to } a\}$. Since W is a proper subset of $N(v)$, X cannot span all faces incident with v . We choose any face contained (not contained) in X and denote it by f (resp. g). Since Π is an embedding, there exists a sub-sequence of edges incident with v that corresponds to a dual path from f to g . But according to (c), all edges incident with X join two faces in X .

(\Leftarrow) We find an embedding Π by forming a face from each component of each cell of P . We establish feasible rotations around each node v in the following way: Let D_v be a directed graph with nodes $N(v)$ such that $uw \in A(D_v)$ if and only if uv and vw are in the same cell of P (i.e., the arcs could be traversed in that order when tracing the face corresponding to v 's component of the cell). A feasible rotation around v corresponds to a Hamiltonian cycle in D_v . We show that D_v is loopy, and hence Hamiltonian by Lemma 1: By construction of D_v , $\mathcal{K}(a)$ is a $K_{k,k}$ for some $k \in \mathbb{N}$, for each $a \in A(D_v)$. By property (b), all nodes of D_v have at least one incoming arc. For disconnected D_v , let W denote the nodes of a single component of D_v , and X the cells that induce $A(D_v[W])$. Then X, v, W contradict property (c). \blacktriangleleft

The above theorem shows that it suffices to optimize over all partitions of arcs into faces. Given a feasible partition (w.r.t. Theorem 2), a corresponding embedding is easily determined in polynomial time following our proof. We can now establish our new model ILP_{Real} , which extends ILP_{Base} with constraints (3). While the former already establishes properties (a) and (b), the latter models property (c): the connectivity of the ‘‘local dual graph’’ D_v around each primal node. Here, index set I corresponds to X from Theorem 2.

$$x(I, v \times_A (N(v) \setminus W)) \geq 1 + x(I, v \times_A W) - 2|W| \quad \forall v \in V, I \subseteq [\bar{f}], W \subsetneq N(v) : W \neq \emptyset \quad (3)$$

Separation. Clearly, it is impractical to add all exponentially many constraints (3) when solving ILP_{Real} . We use a heuristic separation routine to identify a relevant subset of these constraints. For each LP-feasible solution encountered during the solving process, we proceed as follows: For each node v we check if all variables x_a^i of its incident arcs a are integral. If this holds, but the corresponding D_v is disconnected, we found a new violation of (3).

4 Small Faces

The following approach is inspired by the cycle model for the maximum planar subgraph problem [11]. There, a mapping between small faces and short cycles was used to (only) strengthen the LP-relaxation of another, by itself sufficient, model. Thus, it was possible to mostly disregard longer cycles. In our setting we have to be more careful: On the one hand, we need to consider a far wider range of drawings as we embed on surfaces of higher genera. On the other hand, we have to directly adapt the core model itself; to continue to have a sufficient model, we need to *precisely* encode *all*, even very large, faces. We will model ‘‘short’’ faces by new binary y -variables, one for each specific feasible set of arcs. We continue to use the x -variables for *generic*, i.e., ‘‘large’’ faces. On sparse graphs, this yields a reduction of x -variables, as we may drastically decrease the upper bound on the number of generic faces. Both models, ILP_{Pre} and ILP_{Real} , can be extended in this way.

Let \mathcal{C}_χ denote the maximal set of closed walks such that each walk’s length satisfies property χ . Expanding on ILP_{Base} , we parameterize our new model $\text{ILP}_{\text{Base}}^D$ by some $D \geq 2$ and obtain (4a–4g) below. We introduce a new decision variable y_c for each $c \in \mathcal{C}_{\leq D}$. Let $y(a) := \sum_{c \in \mathcal{C}_{\leq D} : a \in c} y_c$ and $\bar{f}_{>d}$ any upper bound on the number of faces with length $> d$.

$$\begin{aligned}
 \max \quad & \sum_{i=1}^{\bar{f}_{>D}} x_i + \sum_{c \in \mathcal{C}_{\leq D}} y_c & (4a) \\
 \text{s.t.} \quad & (D+1)x_i \leq x(i, A) & \forall i \in [\bar{f}_{>D}] & (4b) \\
 & x(a) + y(a) = 1 & \forall a \in A & (4c) \\
 & x(i, \delta_-(v)) = x(i, \delta_+(v)) & \forall i \in [\bar{f}_{>D}], v \in V & (4d) \\
 & \sum_{i=1}^{\bar{f}_{>D}} x_i + \sum_{c \in \mathcal{C}_{\leq D}: |c| > d} y_c \leq \bar{f}_{>d} & \forall d \in \{2, \dots, D\} & (4e) \\
 & x_i \in \{0, 1\}, x_i^a \in \{0, 1\} & \forall i \in [\bar{f}_{>D}], a \in A & (4f) \\
 & y_c \in \{0, 1\} & \forall c \in \mathcal{C}_{\leq D} & (4g)
 \end{aligned}$$

Each arc is contained either in one of the generic faces that each form a set of closed walks (as for ILP_{Base}), or in a closed walk c with dedicated variable y_c , see constraints (4c). Generic faces are large, as required by constraints (4b). Constraints (4d) are essentially (1d). Albeit not required for integral solutions, constraints (4e) enforce the previously implicit upper bound on the total number of faces and bound the number of gradually smaller faces.

Predecessor Model. To obtain $\text{ILP}_{\text{Pre}}^D$, we add equations (2a–2d) to $\text{ILP}_{\text{Base}}^D$, i.e., the same set as for the transition from ILP_{Base} to ILP_{Pre} . Additionally, we require

$$\sum_{c \in \mathcal{C}_{\leq D}: uv, vw \in c} y_c \geq p_{u,w}^v - x([\bar{f}], uv) \quad \forall v \in V, u, w \in N(v). \quad (5)$$

Similar to (2a), this ensures that if an arc uv is contained in a face modeled by a y -variable, the succeeding arc vw has to be contained in the same face.

Realizability Model. We obtain $\text{ILP}_{\text{Real}}^D$ by starting with $\text{ILP}_{\text{Base}}^D$ and adding the following constraints to realize property (c) of Theorem 2.

$$\begin{aligned}
 x(I, \overline{A_W^v}) &\geq 1 + x(I, A_W^v) + \sum_{c \in \mathcal{C}_{\leq D}: c \cap \overline{A_W^v} = \emptyset} |c \cap A_W^v| y_c - 2|W| \\
 &\forall v \in V, I \subseteq [\bar{f}], W \subsetneq N(v) : W \neq \emptyset, A_W^v := v \times_A W, \overline{A_W^v} := v \times_A (N(v) \setminus W)
 \end{aligned} \quad (6)$$

They ensure there is no subset W of arcs at a common node v that is fully assigned to a set (consisting of I and a subset of $\mathcal{C}_{\leq D}$) of face variables that do not have an arc outside of W .

D-Hierarchy: Strength of LP-Relaxations. Clearly $\text{ILP}_{\text{Base}} = \text{ILP}_{\text{Base}}^2$. The value of \bar{f} has a strong influence on the dual bounds obtained by LP-relaxations. Hence, we describe how to determine \bar{f} and $\bar{f}_{>D}$ on general graphs. Let $n := |V(G)|$ and $m := |E(G)|$. Let $f_{\text{UB}}(a, b) := \min\{a, b - \mathbb{1}_{a-b=1 \pmod{2}}\}$, $\bar{f} := f_{\text{UB}}(m - n, \lfloor 2m/3 \rfloor)$, $\bar{f}_{>2} := \bar{f}$, and $\bar{f}_{>d} := \min\{\bar{f}_{>d-1}, \lfloor 2m/(d+1) \rfloor\}$ for $d > 2$. The validity of these bounds follows directly from Euler's formula (assuming non-planar G). We are not aware of any better, general, dual bounds. In the following comparison of LP-relaxations we always assume the above bounds.

► **Lemma 3.** *For every graph G , the LP-relaxation of ILP_{Base} has objective value \bar{f} .*

Proof. The domains (1e) establish \bar{f} as an upper bound. Set $\tilde{x}_a^i = 1/\bar{f}$ and $\tilde{x}^i = 1$ for all $i \in \bar{f}, a \in A$. Clearly \tilde{x} is an LP-feasible solution and achieves the claimed objective. ◀

► **Lemma 4.** *For every graph G , $\text{ILP}_{\text{Base}}^D$ admits an LP-feasible solution with objective value $\bar{f}_{>D}$. If G contains no closed walk of length at most D , this value is optimal.*

Proof. The first claim follows from the LP-feasible solution $\tilde{x}_i^a = 1/\bar{f}_{>D}$ and $\tilde{x}_i = 1$ for all $i \in \bar{f}_{>D}$, $a \in A$, and $\tilde{y} = 0$. When $\mathcal{C}_{\leq D} = \emptyset$, there are no y variables and the domains (4f) bound the objective from above, yielding the second claim. \blacktriangleleft

► **Lemma 5.** *Model $\text{ILP}_{\text{Base}}^{D+1}$ is at least as strong as $\text{ILP}_{\text{Base}}^D$ for any $D \geq 2$.*

Proof. Observe that $\text{ILP}_{\text{Base}}^{D+1}$ generally contains more y - but fewer x -variables than $\text{ILP}_{\text{Base}}^D$. Consider an LP-feasible solution (\hat{x}, \hat{y}) for $\text{ILP}_{\text{Base}}^{D+1}$. We derive an LP-feasible solution (\tilde{x}, \tilde{y}) for $\text{ILP}_{\text{Base}}^D$ that achieves no smaller objective value. For notational simplicity, let $\hat{x}_i = \hat{x}_i^a = 0$ for all $i > \bar{f}_{>D+1}$ and $\beta := \sum_{c \in \mathcal{C}_{=D+1}} \hat{y}_c$. For $\beta = 0$, already (\hat{x}, \hat{y}) , when interpreted for $\text{ILP}_{\text{Base}}^D$, is LP-feasible. Assume $\beta > 0$. Let $\alpha := \bar{f}_{>D} - \sum_{i=1}^{\bar{f}_{>D+1}} \hat{x}_i$ and $\beta_a := \sum_{c \in \mathcal{C}_{=D+1}: a \in c} \hat{y}_c \forall a \in A$. From $\alpha < \beta$ it would follow that $\bar{f}_{>D} < \sum_{i=1}^{\bar{f}_{>D+1}} \bar{f}_{>D+1} \hat{x}_i + \beta$, a direct contradiction of constraint (4e) for $d = D$ in $\text{ILP}_{\text{Base}}^{D+1}$. Thus, $\alpha \geq \beta$. We define (\tilde{x}, \tilde{y}) by $\tilde{x}_i := \hat{x}_i + (1 - \hat{x}_i)\beta/\alpha, \forall i \in [\bar{f}_{>D}]$; $\tilde{x}_i^a := \hat{x}_i^a + (\hat{x}_i - \hat{x}_i^a)\beta_a/\beta, \forall i \in [\bar{f}_{>D}], a \in A$; and $\tilde{y}_c := \hat{y}_c, \forall c \in \mathcal{C}_{\leq D}$. The objective value (4a) for (\tilde{x}, \tilde{y}) in $\text{ILP}_{\text{Base}}^D$ is

$$\begin{aligned} \sum_{i=1}^{\bar{f}_{>D}} \tilde{x}_i + \sum_{c \in \mathcal{C}_{\leq D}} \tilde{y}_c &= \sum_{i=1}^{\bar{f}_{>D}} (\hat{x}_i + (1 - \hat{x}_i)\beta/\alpha) + \sum_{c \in \mathcal{C}_{\leq D}} \hat{y}_c \\ &\stackrel{\text{(by } x_i = 0 \forall i > \bar{f}_{D+1})}{=} \sum_{i=1}^{\bar{f}_{>D+1}} \hat{x}_i + \beta/\alpha \cdot (\bar{f}_{>D} - \sum_{i=1}^{\bar{f}_{>D+1}} \hat{x}_i) + \sum_{c \in \mathcal{C}_{\leq D}} \hat{y}_c \\ &\stackrel{\text{(by def. of } \alpha, \beta)}{=} \sum_{i=1}^{\bar{f}_{>D+1}} \hat{x}_i + \sum_{c \in \mathcal{C}_{\leq D+1}} \hat{y}_c, \end{aligned}$$

i.e., equal to that of (\hat{x}, \hat{y}) in $\text{ILP}_{\text{Base}}^{D+1}$. Assuming constraint (4b) to be violated, we obtain $(D+1)\tilde{x}_i > (D+1)(\tilde{x}_i - \hat{x}_i) + \sum_{a \in A} \hat{x}_i^a \beta_a/\beta$, since $\sum_{a \in A} \beta_a/\beta = D+1$. This implies $(D+1)\tilde{x}_i > \sum_{a \in A} \hat{x}_i^a$, a violation of constraint (4b) already by (\hat{x}, \hat{y}) . Let us show the feasibility of (\tilde{x}, \tilde{y}) w.r.t. constraints (4c) by expanding their left-hand side.

$$\begin{aligned} \tilde{x}(a) + \tilde{y}(a) &= \sum_{i=1}^{\bar{f}_{>D}} (\hat{x}_i^a + (1 - \hat{x}_i)\beta_a/\alpha) + \sum_{c \in \mathcal{C}_{\leq D}: a \in c} \hat{y}_c \\ &\stackrel{\text{(by def. of } \alpha)}{=} \sum_{i=1}^{\bar{f}_{>D+1}} \hat{x}_i^a + \beta_a + \sum_{c \in \mathcal{C}_{\leq D}: a \in c} \hat{y}_c \\ &\stackrel{\text{(by def. of } \beta_a)}{=} \sum_{i=1}^{\bar{f}_{>D+1}} \hat{x}_i^a + \sum_{c \in \mathcal{C}_{\leq D+1}: a \in c} \hat{y}_c \stackrel{\text{(by feasibility of (4c) in } (\hat{x}, \hat{y}))}{=} 1 \end{aligned}$$

Since $\mathcal{C}_{=D+1}$ contains only closed walks, we have $\sum_{u \in N(v)} (\beta_{uv} - \beta_{vu}) = 0$ for all nodes v . Constraints (4d) hold, as we see by expanding their left-hand side:

$$\begin{aligned} \sum_{vu \in A} \tilde{x}_i^{vu} &= \sum_{vu \in A} \hat{x}_i^{vu} + (\tilde{x}_i - \hat{x}_i)/\beta \cdot \sum_{vu \in A} \beta_{vu} \\ &\stackrel{\text{(by (4d) in } \text{ILP}_{\text{Base}}^{D+1} \text{ and the above)}}{=} \sum_{uv \in A} \hat{x}_i^{uv} + (\tilde{x}_i - \hat{x}_i)/\beta \cdot \sum_{uv \in A} \beta_{uv} = \sum_{uv \in A} \hat{x}_i^{uv} \end{aligned}$$

Constraints (4e) maintain their slack, as the first term increases by $\sum_{i=1}^{\bar{f}_{>D}} (\tilde{x}_i - \hat{x}_i) = \beta$ while the second decreases by β . Clearly, $\tilde{x}_i \geq \hat{x}_i$ and $\tilde{x}_i^a \geq \hat{x}_i^a$. By $\alpha \geq \beta$ we have $\tilde{x}_i \leq 1$. By (4c) we have $\hat{x}_i^a + \beta_a \leq 1$. Thus, $\tilde{x}_i^a > 1$ would imply $\tilde{x}_i - \hat{x}_i > \beta$. Clearly, we keep $0 \leq \tilde{y}_c \leq 1$. \blacktriangleleft

► **Theorem 6.** *Model $\text{ILP}_{\text{Base}}^{D+1}$ is stronger than $\text{ILP}_{\text{Base}}^D$ for any $D \geq 2$.*

Proof. Restricting ourselves to dense graphs of girth $> D+1$, the claim immediately follows from Lemmata 3–5. An example of such graphs are the complete graphs on D nodes, where we subdivide each edge D times. They have girth $3(D+1)$ and are dense enough such that the respective bounds differ: $f_{>D} > f_{>D+1}$. We note that there are also dense graphs with high girth that do not allow any general preprocessing techniques. \blacktriangleleft

5 Additional Tuning (“Add-Ons”)

In addition to the new models described above, there is a set of supplemental constraints that may be applied to several of these models. We discuss them in alphabetical order.

Arc-Face. We may require the below trivial constraints explicitly.

$$x_i \geq x_i^a \quad \forall a \in A, i \in [\bar{f}] \quad (7)$$

Branching Rule. To facilitate the fast generation of strong primal bounds, we may initially restrict the solution space to explicitly modeled faces, e.g., by branching on

$$\sum_{i \in \bar{f}_{>D}} x_i \stackrel{?}{=} 0. \quad (8)$$

deg-3-Model. There are only two possible rotations around any degree-3 node v . In ILP_{Pre} , this can be modeled by a single binary variable for v and alternative constraints, partially replacing (2a–2d), as discussed in [2]. The same holds for $\text{ILP}_{\text{Pre}}^D$, and we use this improvement in our benchmarks.

Long Faces. In several cases we can establish lower bounds on the length of faces modeled by the x -variables. Let $s(v, w)$ denote the length of the shortest path between nodes v and w .

► **Lemma 7.** *For arcs uv, wx that traverse the same face f , we have $s(v, w) + s(x, u) + 2 \leq |f|$.*

Proof. Tracing any such face f yields a path from v to w that neither contains arc uv (it may contain arc vu) nor arc vx . Similarly, an arc-disjoint path from x to u must exist in f . The total length of these paths is lower bounded by $s(v, w) + s(x, u)$. ◀

► **Lemma 8.** *Any face with a singular edge contains at least eight arcs and this is tight.*

Proof. Let uv denote the edge that is traversed in both directions when tracing face f . If tracing f would additionally yield a path from u to v that does not contain uv , the tracing would similarly yield a path from v to u that does not contain vu . This contradicts the assumption since f would contain two oppositely directed, closed walks that form two separate faces. Hence, there exist two arc-disjoint closed walks on the boundary of f , one for each node u, v . Since there are no deg-1 nodes in a biconnected graph, any subcycle in a face requires at least three arcs and the claim follows. Considering a genus-1 embedding of the K_4 we can see that it indeed contains such a face of length eight. ◀

► **Lemma 9.** *Any face with a singular node contains at least six arcs and this is tight.*

Proof. If the face f also traverses an edge twice, the bound follows from Lemma 8. Otherwise, the doubly traversed node has at least four arcs in f , belonging to pairwise different edges, and hence four incident nodes. A closed walk on this $K_{1,4}$ requires at least two additional arcs and the claimed bound follows. A face of length six can be observed in a genus-1 embedding of the following graph: Take two copies of the K_5 , remove one edge each, join the graphs by identifying two deg-3 nodes, and add a new edge between the remaining two deg-3 nodes. ◀

Let $\ell_{uv,wx} := \max\{s(v, w) + s(x, u) + 2, 6 \cdot \mathbf{1}_{k=3}, 8 \cdot \mathbf{1}_{k=2}\}$ with $k := |\{u, v, w, x\}|$. Lemmata 7–9 yield the following constraints. When using them in our benchmarks, we separate them.

$$\ell_{uv,wx} (x_i^{uv} + x_i^{wx} - 1) \leq x(i, A) \quad \forall i \in [\bar{f}_{>D}], uv, wx \in A : uv \neq wx \quad (9)$$

Objective Parity. All above ILPs maximize the number f of faces and deduce γ via Euler’s formula $n + f - m = 2 - 2\gamma$. Thus the parity of f is fixed. This gives room for improved bounding and cutting by the ILP solver. Using a new variable $z \in \mathbb{N}$ we may demand

$$(m - n \bmod 2) + 2z = \sum_{i=1}^{\bar{f} > D} f_i + \sum_{c \in \mathcal{C}_{\leq D}} y_c. \quad (10)$$

Symmetry Breaking. For ILP_{Pre} it was observed in [2] that symmetry breaking does not seem to pay off. However, ILP_{Pre} only solves genus-1 instances in practice (see below). Symmetry breaking may hinder heuristics from identifying trivially optimal solutions, but may be beneficial for harder instances. The approach in [2] enforces that face i has at most as many arcs as face $i + 1$. However, there are typically many faces with the same length in a low-genus embedding. We consider breaking symmetries by restricting the set of faces that may contain a given arc. Let \prec denote an arbitrary but fixed order on the arc set A .

$$y(a) + x(\{1, \dots, \ell\}, a) \geq 1 - x(\ell, \{a' \in A : a' \prec a\}) \quad \forall \ell \in [\bar{f}], a \in A \quad (11)$$

These constraints ensure that any arc is contained either in an explicitly modeled closed walk or in the lowest-indexed face that it can be placed into. For ILP_{Pre} and ILP_{Real} , i.e., when there are no explicitly modeled closed walks, we simply set $y(a) = 0$.

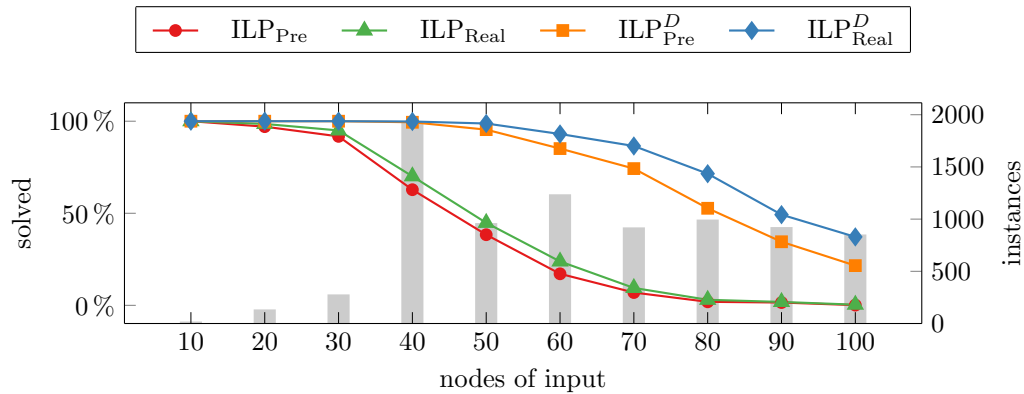
6 Experiments

All algorithms are implemented in C++, compiled with gcc 6.3.0, and use the OGDF (snapshot 2018-03-28) [9]. We use SCIP 6.0.0 [16] for solving ILPs, with CPLEX 12.8.0 as the underlying LP solver. Each 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 10 minutes and a memory limit of 8 GB per computation. All instances and results, giving runtime and genus (if solved), are available for download at <http://tcs.uos.de/research/min-genus>. In our experiments, we increase parameter D – separately on each graph – until we obtain at least 1000 y -variables. As they are not required for integral solutions, we omit constraints (4e) by default.

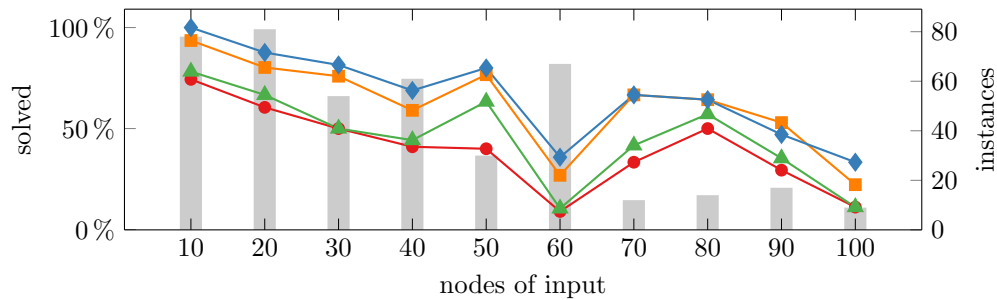
Instance Sets. We consider the 423 and 8249 non-planar graphs of the two established real-world sets NORTH [28] and ROME [14], respectively. In addition, we use the set of 600 EXPANDER graphs, as established in [10, 11] for a related non-planarity measure (skewness): there are 20 graphs for each feasible parameterization $(|V(G)|, \Delta) \in \{10, 20, 30, 50, 100\} \times \{4, 6, 10, 20, 40\}$, where Δ denotes the node degree.

Discussion of SAT-based algorithms. In [2], the SAT-based approach was faster than the ILP-based one. However, we do not need to directly compare with it.

Both previous approaches solve only instances with genus ≤ 1 in practice. Since the respective dual bound is trivially given by planarity testing (and enforced in all previous models), the runtime difference can be attributed to the SAT-solver quickly finding a satisfying solution. In contrast, standard primal heuristics of ILP-solvers are weaker, and the comparably time-consuming LP-relaxations are rarely profitable. However, w.r.t. success-rate, SAT is only marginally in the lead, if at all: on the ROME graphs, the ILP and SAT solve 2595 and 2667 instances, respectively. For NORTH, “the success-rates of both approaches are [...] comparable” [2].



(a) Solved ROME graphs by number of nodes.



(b) Solved NORTH graphs by number of nodes.

■ **Figure 1** Detailed success-rates of algorithms on established benchmark sets. We provide the relative number of solved instances over the number of nodes, clustered to the nearest multiple of 10. The gray bars denote the number of instances in each cluster.

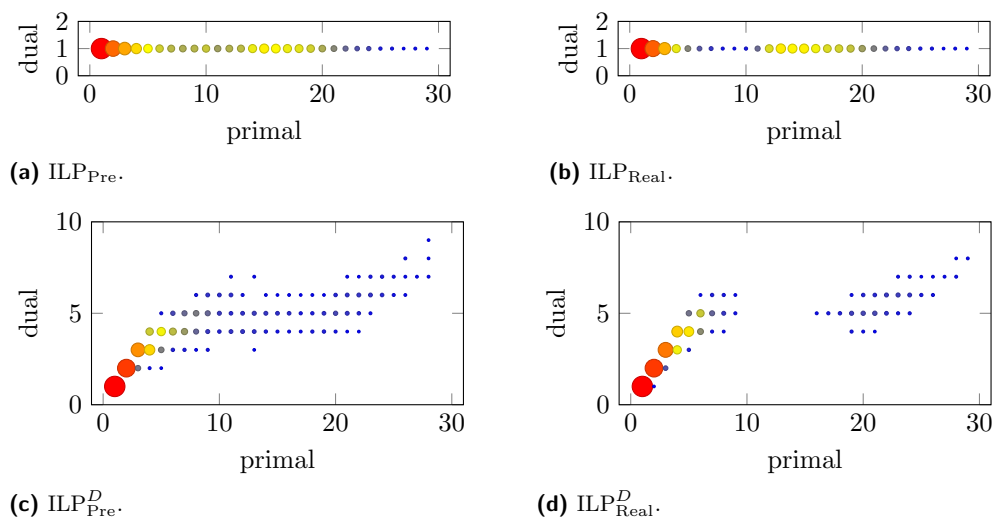
Since we can neither employ separation nor LP-relaxations in the setting of SAT-solvers, there also is no immediate way of using our strengthening results for SAT-based algorithms. We will see that the new ILP variants clearly dominate the SAT-based variants; e.g., we solve up to 6797 ROME instances.

Results. The experiments confirm that our new model is not only theoretically stronger but also better in practice: Using $\text{ILP}_{\text{Real}}^D$, we are now able to solve 82% instead of just 28% of the ROME graphs, cf. Table 1. Depending on the instance set, we achieve an average speed-up of factor 82 to 248. In [2], only graphs with genus 1 (and not all of them) could be solved. Surprisingly, and in contrast to the observations made in [2], the deg 3-model does not perform better than the respective base variant: SCIP’s built-in preprocessing reduces the variable space to essentially the same dimension as obtained when manually applying the deg 3-model (while possibly retaining some additional information that helps in the solving process). Also somewhat to our surprise, none of the add-ons (8–11) pay off *reliably*.

Taking a closer look at the number of solved instances (Figure 1 and Table 2), we see that – on average – $\text{ILP}_{\text{Real}}^D$ is superior to all other variants for any graph size. We now solve real-world instances with non-trivial dual bounds, i.e., when the genus is > 1 , e.g., we have solved a genus-7 instance on ROME and even a genus-21 instance on NORTH. We see very clearly, in particular on ROME, that we may order the models ILP_{Pre} , ILP_{Real} , $\text{ILP}_{\text{Pre}}^D$, $\text{ILP}_{\text{Real}}^D$ by increasing success rate. This means that, independent on whether we apply the small-faces model extension or not, the realizability model is more successful than

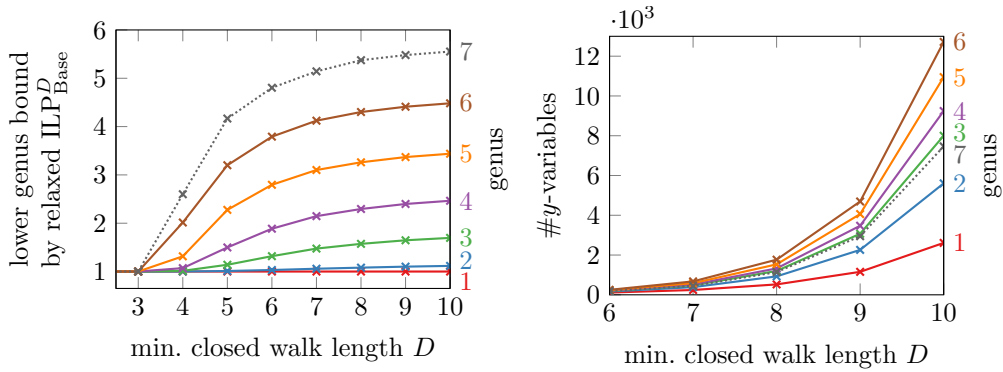
■ **Table 1** Average success-rate s and runtime t on each instance set. Considering the runtime, we restrict the instances to those solved by all variants. Of the graphs solved by ILP_{Pre} , all variants solved at least 94% (ROME), 99% (NORTH), and 100% (EXPANDER). Particularly, algorithms based on ILP_{Base}^D always solved all of the instances solved by ILP_{Pre} .

	ROME		NORTH		EXPANDER	
	s [%]	t [s]	s [%]	t [s]	s [%]	t [s]
ILP_{Pre}	27.79	190.23	45.86	139.09	5.26	58.29
$ILP_{Pre} + \text{deg } 3$	27.94	186.31	47.52	111.50	5.26	58.53
ILP_{Real}	31.85	70.57	50.83	25.44	5.53	11.33
ILP_{Pre}^D	73.08	2.92	67.14	12.21	17.37	2.24
$ILP_{Pre}^D + \text{deg } 3$	68.09	3.86	65.01	12.47	17.37	2.25
ILP_{Real}^D	81.65	0.91	73.52	7.26	23.95	0.95
$ILP_{Real}^D + \text{branch rule (8)}$	76.41	0.86	73.05	7.29	21.05	0.80
$ILP_{Real}^D + \text{all symmetries (11)}$	81.56	0.91	73.52	7.27	23.95	0.95
$ILP_{Real}^D + \text{sepa. symmetries (11)}$	81.59	0.91	73.76	7.26	23.95	0.94
$ILP_{Real}^D + \text{sepa. long faces (9)}$	81.16	0.95	72.81	7.27	22.89	0.81
$ILP_{Real}^D + \text{all \#faces cons. (4e)}$	81.57	0.75	74.00	6.79	23.68	0.86
$ILP_{Real}^D + \text{sepa. \#faces cons. (4e)}$	81.60	1.13	74.00	7.02	23.95	1.05
$ILP_{Real}^D + \text{parity model (10)}$	82.33	1.25	73.29	1.11	22.89	1.68
$ILP_{Real}^D + \text{all arc-face cons. (7)}$	75.43	0.98	71.39	7.35	18.95	0.71
$ILP_{Real}^D + \text{sepa. arc-face cons. (7)}$	81.71	1.48	73.76	7.28	23.42	0.76
$ILP_{Real}^D + (10) + \text{sepa. (4e,7)}$	82.40	1.37	74.00	1.20	22.63	1.65
$ILP_{Real}^D + (8) + (10) + \text{sepa. (4e,7)}$	78.07	1.19	75.65	1.18	21.58	1.62



■ **Figure 2** Final primal vs. dual bounds on the genus, generated by algorithmic variants on ROME (without any add-ons). Color and size indicate the number of instances with the respective bounds. We note that these bounds do not apply to the values of the formal objective value, i.e., the number of attained faces, but to the genus, which allows a more sensible comparison. Note that without the small faces extension, neither ILP_{Pre} nor ILP_{Real} obtains lower bounds > 1 (i.e., only trivial ones).

30:12 Stronger ILPs for the Graph Genus Problem



(a) genus bound of LP-relaxation.

(b) number of generated y -variables.

■ **Figure 3** Average values of ILP_{Base}^D on the solved ROME graphs, depending on the maximum length D of explicitly modeled cycles and the genus of the graph. Note that we only solved one instance with genus 7 on ROME.

■ **Table 2** Number of solved instances in the EXPANDER set for selected variants without add-ons.

# nodes	10		20			30		≥ 50
	4	6	4	6	10	4	6-20	4-40
ILP_{Pre}	20	0	0	0	0	0	0	0
ILP_{Real}	20	1	0	0	0	0	0	0
ILP_{Pre}^D	20	20	20	0	0	6	0	0
ILP_{Real}^D	20	20	20	18	0	13	0	0

the predecessor model. The most progress, however, is achieved by activating the small-faces model extension ILP_{Base}^D . As the shapes of the success-rate curves demonstrate, it benefits both underlying models roughly equally. In particular, we see (cf. also Figure 2 which shows the final bounds of our core variants on ROME) that even ILP_{Real} , like ILP_{Pre} , can only solve genus 1 instances. This is in accordance with Lemma 3, i.e., that the LP-relaxation of ILP_{Base} always yields value \bar{f} . Nonetheless, the success-rates 46% and 51% on NORTH for ILP_{Pre} and ILP_{Real} , respectively, demonstrate that ILP_{Pre} is far from solving all toroidal instances. More complex instances require the small-faces extension ILP_{Base}^D . This is also reflected by the root relaxations of ILP_{Base}^D for different values of D , cf. Figure 3. Consistent with theory, increasing the minimum length D leads to stronger LP-relaxations also in practice, but may drastically increase the number of variables. Interestingly, generating only triangles, i.e., $D = 3$, yields only a very slight increase on the average dual bound compared to ILP_{Base} on ROME, possibly caused by the graphs' sparsity.

Genera in Graph Theory. Our new approach allows us to confirm results from literature, all with non-trivial dual bounds: In 2015, the circulants of genus ≤ 2 were characterized [12]. Thereby, the authors need to show that 12 specific graphs have genus ≥ 3 . For these arguments alone, they require about nine pages, supplemented by several hours of computation. Using ILP_{Real}^D , we are able to confirm these results (and compute the respective genera) in a matter of seconds without employing any graph-specific theory. Before, using ILP_{Pre} , the arguably hardest case $C_{11}(1, 2, 4)$ required 180 hours [2]. In 2005, a full paper was dedicated to showing that the Gray graph has genus 7 [24]. Our tool confirms this result within 42 hours. Similarly, we confirm a result from 1989 [4] in 250 seconds: the group that is the semidirect product of \mathbb{Z}_9 with \mathbb{Z}_3 has genus 4 (the genus of group Γ is the smallest genus of a Caley graph of Γ).

7 Conclusion

We have presented novel ILP models for the graph genus problem, proved their theoretical strength, the existence of a hierarchy of ever stronger LP-relaxations, and positively evaluated them in practice, e.g., by solving 82% instead of the previous 28% of the ROME instances. We are now able to solve real-world instances with genera up to 21. This is in stark contrast to the previous models that – on the same set of instances – succeeded only on toroidal graphs.

It remains open whether even stronger models can be found by a more careful examination of the face structure. What additional properties of the embedding may be modeled? Is it possible to better exploit singular nodes or edges, particularly when they are adjacent? Further, we expect that our algorithms would benefit from strong primal heuristics but we are not aware of any general such algorithms. Currently, optimal dual bounds are often identified long before an optimal solution is found.

References

- 1 Amariah Becker, Philip N. Klein, and David Saulpic. A Quasi-Polynomial-Time Approximation Scheme for Vehicle Routing on Planar and Bounded-Genus Graphs. In *ESA 2017*, pages 12:1–12:15, 2017. doi:10.4230/LIPIcs.ESA.2017.12.
- 2 Stephan Beyer, Markus Chimani, Ivo Hedtke, and Michal Kotrbčík. A Practical Method for the Minimum Genus of a Graph: Models and Experiments. In *SEA 2016*, LNCS 9685, pages 75–88, 2016. doi:10.1007/978-3-319-38851-9_6.
- 3 C. Paul Bonnington and Tomaz Pisanski. On the orientable genus of the cartesian product of a complete regular tripartite graph with an even cycle. *Ars Comb.*, 70, 2004.
- 4 Matthew G. Brin, David E. Rauschenberg, and Craig C. Squier. On the genus of the semidirect product of \mathbb{Z}_9 by \mathbb{Z}_3 . *Journal of Graph Theory*, 13(1):49–61, 1989. doi:10.1002/jgt.3190130108.
- 5 Matthew G. Brin and Craig C. Squier. On the Genus of $\mathbb{Z}_3 \times \mathbb{Z}_3 \times \mathbb{Z}_3$. *Eur. J. Comb.*, 9(5):431–443, 1988. doi:10.1016/S0195-6698(88)80002-7.
- 6 Chandra Chekuri and Anastasios Sidiropoulos. Approximation Algorithms for Euler Genus and Related Problems. In *FOCS 2013*, pages 167–176, 2013. doi:10.1109/FOCS.2013.26.
- 7 Jianer Chen. A Linear-Time Algorithm for Isomorphism of Graphs of Bounded Average Genus. *J. Disc. Math.*, 7(4):614–631, 1994. doi:10.1137/S0895480191196769.
- 8 Jianer Chen, Iyad A. Kanj, Ljubomir Perkovic, Eric Sedgwick, and Ge Xia. Genus characterizes the complexity of certain graph problems: Some tight results. *J. Comput. Syst. Sci.*, 73(6):892–907, 2007. doi:10.1016/j.jcss.2006.11.001.
- 9 Markus Chimani, Carsten Gutwenger, Mike Juenger, 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>.
- 10 Markus Chimani, Ivo Hedtke, and Tilo Wiedera. Exact Algorithms for the Maximum Planar Subgraph Problem: New Models and Experiments. In *SEA 2018*, pages 22:1–22:15, 2018. doi:10.4230/LIPIcs.SEA.2018.22.
- 11 Markus Chimani and Tilo Wiedera. Cycles to the Rescue! Novel Constraints to Compute Maximum Planar Subgraphs Fast. In *ESA 2018*, LIPIcs 112, pages 19:1–19:14, 2018. doi:10.4230/LIPIcs.ESA.2018.19.
- 12 Marston Conder and Ricardo Grande. On Embeddings of Circulant Graphs. *Electr. J. Comb.*, 22(2):P2.28, 2015. URL: <http://www.combinatorics.org/ojs/index.php/eljc/article/view/v22i2p28>.
- 13 Erik D. Demaine, MohammadTaghi Hajiaghayi, and Dimitrios M. Thilikos. The Bidimensional Theory of Bounded-Genus Graphs. *Disc. Math.*, 20(2):357–371, 2006. doi:10.1137/040616929.

- 14 Giuseppe Di Battista, Ashim Garg, Giuseppe Liotta, Roberto Tamassia, Emanuele Tassinari, and Francesco Vargiu. An Experimental Comparison of Four Graph Drawing Algorithms. *Comput. Geom.*, 7:303–325, 1997. doi:10.1016/S0925-7721(96)00005-3.
- 15 John A. Ellis, Hongbing Fan, and Michael R. Fellows. The dominating set problem is fixed parameter tractable for graphs of bounded genus. *J. Algorithms*, 52(2):152–168, 2004. doi:10.1016/j.jalgor.2004.02.001.
- 16 Ambros Gleixner, Michael Bastubbe, Leon Eifler, Tristan Gally, Gerald Gamrath, Robert Lion Gottwald, Gregor Hendel, Christopher Hojny, Thorsten Koch, Marco E. Lübbecke, Stephen J. Maher, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Franziska Schlösser, Christoph Schubert, Felipe Serrano, Yuji Shinano, Jan Merlin Viernickel, Matthias Walter, Fabian Wegscheider, Jonas T. Witt, and Jakob Witzig. The SCIP Optimization Suite 6.0. Technical report, Optimization Online, July 2018. URL: http://www.optimization-online.org/DB_HTML/2018/07/6692.html.
- 17 David A. Hoelzeman and Saïd Bettayeb. On the genus of star graphs. *IEEE Transactions on Computers*, 43(6):755–759, 1994. doi:10.1109/12.286309.
- 18 Mark Jungerman and Gerhard Ringel. The genus of the n -octahedron: Regular cases. *J. Graph Theory*, 2(1):69–75, 1978. doi:10.1002/jgt.3190020109.
- 19 K. Kawarabayashi, B. Mohar, and B. Reed. A Simpler Linear Time Algorithm for Embedding Graphs into an Arbitrary Surface and the Genus of Graphs of Bounded Tree-Width. In *FOCS 2008*, pages 771–780, 2008. doi:10.1109/FOCS.2008.53.
- 20 Ken-ichi Kawarabayashi and Anastasios Sidiropoulos. Beyond the Euler Characteristic: Approximating the Genus of General Graphs. In *STOC 2015*, pages 675–682, 2015. doi:10.1145/2746539.2746583.
- 21 Michal Kotrbčík and Tomaz Pisanski. Genus of the Cartesian Product of Triangles. *Electr. J. Comb.*, 22(4):P4.2, 2015. URL: <http://www.combinatorics.org/ojs/index.php/eljc/article/view/v22i4p2>.
- 22 Valentas Kurauskas. On the genus of the complete tripartite graph $K_{n,n,1}$. *Disc. Math.*, 340(3):508–515, 2017. doi:10.1016/j.disc.2016.09.017.
- 23 Yury Makarychev, Amir Nayyeri, and Anastasios Sidiropoulos. A Pseudo-approximation for the Genus of Hamiltonian Graphs. In *APPROX 2013*, pages 244–259, 2013. doi:10.1007/978-3-642-40328-6_18.
- 24 Dragan Marusic, Tomaz Pisanski, and Steve Wilson. The genus of the GRAY graph is 7. *Eur. J. Comb.*, 26(3-4):377–385, 2005. doi:10.1016/j.ejc.2004.01.015.
- 25 Bojan Mohar. A Linear Time Algorithm for Embedding Graphs in an Arbitrary Surface. *J. Disc. Math.*, 12(1):6–26, 1999. doi:10.1137/S089548019529248X.
- 26 Bojan Mohar, Tomaz Pisanski, Martin Skoviera, and Arthur T. White. The cartesian product of three triangles can be embedded into a surface of genus 7. *Disc. Math.*, 56(1):87–89, 1985. doi:10.1016/0012-365X(85)90197-9.
- 27 Wendy Myrvold and William Kocay. Errors in graph embedding algorithms. *J. Comp. and Sys. Sciences*, 77(2):430–438, 2011. doi:10.1016/j.jcss.2010.06.002.
- 28 Stephen C. North. 5114 directed graphs, 1995. Manuscript.
- 29 Marcin Pilipczuk, Michal Pilipczuk, Piotr Sankowski, and Erik Jan van Leeuwen. Network Sparsification for Steiner Problems on Planar and Bounded-Genus Graphs. *ACM Trans. Algorithms*, 14(4):53:1–53:73, 2018. doi:10.1145/3239560.
- 30 Gerhard Ringel. Das Geschlecht des vollständigen paaren Graphen. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 28(3):139–150, 1965. doi:10.1007/BF02993245.
- 31 Gerhard Ringel. On the genus of the graph $K_n \times K_2$ or the n -prism. *Disc. Math.*, 20:287–294, 1977. doi:10.1016/0012-365X(77)90067-X.
- 32 Gerhard Ringel and J. W. T. Youngs. SOLUTION OF THE HEAWOOD MAP-COLORING PROBLEM. *PNAS USA*, 60(2):438–445, 1968. doi:10.1073/pnas.60.2.438.

- 33 Gerhard Ringel and J. W. T. Youngs. Das Geschlecht des vollständigen dreifärbbaren Graphen. *Commentarii Mathematici Helvetici*, 45(1):152–158, 1970. doi:10.1007/BF02567322.
- 34 Neil Robertson and Paul D. Seymour. Graph minors. VIII. A Kuratowski theorem for general surfaces. *J. Comb. Theory, Ser. B*, 48(2):255–288, 1990. doi:10.1016/0095-8956(90)90121-F.
- 35 Carsten Thomassen. The graph genus problem is NP-complete. *J. Algorithms*, 10(4):568–576, 1989. doi:10.1016/0196-6774(89)90006-0.
- 36 Carsten Thomassen. Embeddings of graphs with no short noncontractible cycles. *J. Comb. Theory, Series B*, 48(2):155–177, 1990. doi:10.1016/0095-8956(90)90115-G.
- 37 Arthur T. White. The genus of the complete tripartite graph $K_{m,n,n}$. *J. Comb. Theory*, 7(3):283–285, 1969. doi:10.1016/S0021-9800(69)80027-X.

Complexity of C_k -Coloring in Hereditary Classes of Graphs

Maria Chudnovsky

Princeton University, Princeton, NJ 08544, USA
mchudnov@math.princeton.edu

Shenwei Huang¹

College of Computer Science, Nankai University, Tianjin 300350, China
shenwei Huang@nankai.edu.cn

Paweł Rzażewski

Faculty of Mathematics and Information Science, Warsaw University of Technology, Warsaw, Poland
p.rzazewski@mini.pw.edu.pl

Sophie Spirkl

Rutgers University, Piscataway, NJ 08854, USA
sophiespirkl@gmail.com

Mingxian Zhong

Lehman College, CUNY, Bronx, NY 10468, USA
mingxian.zhong@lehman.cuny.edu

Abstract

For a graph F , a graph G is F -free if it does not contain an induced subgraph isomorphic to F . For two graphs G and H , an H -coloring of G is a mapping $f : V(G) \rightarrow V(H)$ such that for every edge $uv \in E(G)$ it holds that $f(u)f(v) \in E(H)$. We are interested in the complexity of the problem H -COLORING, which asks for the existence of an H -coloring of an input graph G . In particular, we consider H -COLORING of F -free graphs, where F is a fixed graph and H is an odd cycle of length at least 5. This problem is closely related to the well known open problem of determining the complexity of 3-COLORING of P_t -free graphs.

We show that for every odd $k \geq 5$ the C_k -COLORING problem, even in the precoloring-extension variant, can be solved in polynomial time in P_9 -free graphs. On the other hand, we prove that the extension version of C_k -COLORING is NP-complete for F -free graphs whenever some component of F is not a subgraph of a subdivided claw.

2012 ACM Subject Classification Mathematics of computing \rightarrow Graph algorithms

Keywords and phrases homomorphism, hereditary class, computational complexity, forbidden induced subgraph

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.31

Funding *Maria Chudnovsky*: Supported by NSF grant DMS-1763817. This material is based upon work supported in part by the U. S. Army Research Laboratory and the U. S. Army Research Office under grant number W911NF-16-1-0404.

Shenwei Huang: This research is supported by the National Natural Science Foundation of China (11801284).

Sophie Spirkl: This material is based upon work supported by the National Science Foundation under Award No. DMS1802201.

Acknowledgements We are grateful to anonymous reviewers for their comments that helped improve the presentation of the paper.

¹ Corresponding author



1 Introduction

For graphs G and H , a *homomorphism from G to H* is a mapping $f : V(G) \rightarrow V(H)$ such that $f(u)f(v) \in E(H)$ for every edge $uv \in E(G)$. It is straightforward to see that if H is a complete graph with k vertices, then every homomorphism to H is in fact a k -coloring of G (and vice versa). This shows that homomorphisms can be seen as a generalization of graph colorings. Because of that, a homomorphism to H is often called an *H -coloring*, and vertices of H are called *colors*. We also say that G is *H -colorable* if G has an H -coloring.

In what follows, the target graph H is always fixed. We are interested in the complexity of the H -COLORING problem, which asks whether the input graph G has an H -coloring.

1.1 Complexity of variants of H -Coloring

Since H -COLORING is a generalization of k -COLORING, it is natural to try to extend results for k -COLORING to target graphs H which are not complete graphs. For example, it is well-known that k -COLORING enjoys a complexity dichotomy: it is polynomial-time solvable if $k \leq 2$, and NP-complete otherwise. The complexity dichotomy for H -COLORING was described by Hell and Nešetřil in their seminal paper [23]: they proved that the problem is polynomial-time solvable if H is bipartite, and NP-complete otherwise.

Since then, there have been numerous studies on variants of H -COLORING. Our main focus will be on the H -PRECOLORING EXTENSION problem, in which we are given a triple (G, W, h) , where G is a graph, W is a subset of $V(G)$, and h is a mapping from W to $V(H)$. The problem is to decide if h can be extended to an H -coloring of G , that is, if there is an H -coloring f of G such that $f|_W = h$.

Note that this problem is closely related to the LIST H -COLORING problem, where the input consists of a graph G with an *H -list assignment*, which is a function $L : V(G) \rightarrow 2^{V(H)}$ that assigns a subset of $V(H)$ (called *list*) to each vertex of G . We ask if there is an H -coloring f of G such that $f(v) \in L(v)$ for each $v \in V(G)$. In such a case we say that (G, L) is *H -colorable* and f is an *H -coloring of (G, L)* . Clearly H -PRECOLORING EXTENSION can be seen as a restriction of LIST H -COLORING, in which every list is either a singleton, or contains all vertices of H . This is the reason why it is sometimes called *one-or-all list homomorphism (coloring) problem* [13].

In general, variants of H -COLORING can be seen in a wider context of Constraint Satisfaction Problems (CSP). A full complexity dichotomy for this family of problems has been a long-standing open question, known as the *CSP dichotomy conjecture* of Feder and Vardi [15]. After a long series of partial results, the problem was finally solved very recently, independently by Bulatov [5] and by Zhuk [33].

A natural approach in dealing with computationally hard problems is to consider restricted instances, in hope to understand the boundary between easy and hard cases. For example, it is known that H -COLORING can be solved in polynomial time for perfect graphs, because it suffices to test whether $\omega(G) > \omega(H)$, which can be done in $O(|V(G)|^{|V(H)|})$ time. If $\omega(G) > \omega(H)$, then the answer is no, as there is no way to map the largest clique of G to H . Otherwise the answer is yes, since $\omega(G)$ -coloring of G can be translated to a homomorphism of G to the largest clique of H , and thus to H . The situation changes when we consider the more general setting of H -PRECOLORING EXTENSION and LIST H -COLORING. For any fixed graph H , LIST H -COLORING (and thus H -PRECOLORING EXTENSION and H -COLORING) can be solved in polynomial time for input graphs with bounded tree-width. Combining this with an observation that any graph with a clique larger than $\omega(H)$ has no H -coloring, we obtain polynomial-time algorithms for chordal graphs [14]. For permutations graphs, LIST

H -COLORING can also be solved in polynomial time via a recursive branching algorithm [11]. For bipartite input graphs, however, 3-PRECOLORING EXTENSION (i.e., K_3 -PRECOLORING EXTENSION) is already NP-complete [29]. Other restricted inputs have been studied too, e.g. bounded-degree graphs [16]. For more results on graph homomorphisms, we refer to the monograph by Hell and Nešetřil [22].

1.2 Graphs with forbidden induced subgraphs

A rich family of restricted graph classes comes from forbidding some small substructures. For graphs G and F , we say that G *contains* F if F is an induced subgraph of G . By F -free graphs we mean the class of graphs that do not contain F . Note that this class is *hereditary*, that is, it is closed under taking induced subgraphs.

The complexity of k -COLORING for hereditary graph classes has received much attention in the past two decades and significant progress has been made. Of particular interest is the class of F -free graphs for a fixed graph F . For any fixed $k \geq 3$, the k -COLORING problem remains NP-complete for F -free graphs whenever F is not a linear forest (a collection of disjoint paths) [25, 31]. The simplest linear forests are paths, and the complexity of k -COLORING in P_t -free graphs has been studied by many researchers.

On the positive side, Hoàng, Kamiński, Lozin, Sawada, and Shu [24] gave a recursive algorithm showing that k -COLORING can be solved in polynomial time for P_5 -free graphs for any fixed k . Bonomo, Chudnovsky, Maceli, Schaudt, Stein, and Zhong [4] showed that 3-COLORING can be solved in polynomial time in P_7 -free graphs. Moreover, very recently, Chudnovsky, Spirkl, and Zhong proved that 4-COLORING is polynomial-time solvable in P_6 -free graphs [7, 8, 9].

On the negative side, Woeginger and Sgall [32] demonstrated the NP-completeness of 5-COLORING for P_8 -free graphs and 4-COLORING for P_{12} -free graphs. Later on, these NP-completeness results were improved by various researchers and the strongest result is due to Huang [26] who proved that 4-COLORING is NP-complete for P_7 -free graphs and 5-COLORING is NP-complete for P_6 -free graphs. These results settle the complexity of k -COLORING for P_t -free graphs for all pairs (k, t) , except for the complexity of 3-COLORING for P_t -free graphs when $t \geq 8$. Interestingly, all polynomial-time results carry over to the list variant, except for the case of LIST 4-COLORING of P_6 -free graphs, which was shown to be NP-complete by Golovach, Paulusma, and Song [18]. We refer the reader to the survey by Golovach, Johnson, Paulusma, and Song [17] for more information about coloring graphs with forbidden subgraphs.

Understanding the complexity of 3-COLORING in P_t -free graphs seems a hard problem – on the one hand, algorithms even for small values of t are difficult to construct, and on the other hand all our hardness reductions appear to introduce long induced paths. Let us mention a problem whose complexity is equally elusive: INDEPENDENT SET. Alekseev [1] observed that INDEPENDENT SET is NP-complete in F -free graphs whenever F is not a path or a subdivided claw. For P_t -free graphs, polynomial-time algorithms are known only for small values of t : currently, the best result is the recent polynomial-time algorithm for P_6 -free graphs by Grzesik, Klimošova, Pilipczuk, and Pilipczuk [20, 21]. On the other hand, the problem is not known to be NP-hard for any fixed t .

A natural question to ask is if the similar behavior of 3-COLORING and INDEPENDENT SET in P_t -free graphs is a part of a more general phenomenon. Recently, Groenland, Okrasa, Rzażewski, Scott, Seymour, and Spirkl [19] shed some light on this question by showing that if H does not contain two vertices with two common neighbors, then a very general, weighted variant of H -COLORING can be solved in time $2^{O(\sqrt{tn \log n})}$ for P_t -free graphs. Clearly K_3

does not have two vertices with two common neighbors. Moreover, INDEPENDENT SET can be expressed as a weighted homomorphism to $\textcircled{1}-\textcircled{1}$, which has the same property, and thus, for every t , both 3-COLORING and INDEPENDENT SET can be solved in subexponential time in P_t -free graphs (we note that a subexponential algorithm for INDEPENDENT SET in P_t -free graphs was known before [2]). This implies that if one attempts to prove NP-completeness of any of these problems in P_t -free graphs, then, assuming the Exponential Time Hypothesis [27, 28], such a reduction should be sufficiently complicated to introduce at least a quadratic blow-up of the instance.

In this paper, we study the complexity of variants of H -COLORING when H is an odd cycle of length at least five. Note that by the result of Groenland *et al.* [19], this problem can be solved in subexponential time in P_t -free graphs. We are interested in better classification of polynomial and NP-hard cases.

1.3 Our contribution

The contribution of the paper is twofold: First, we show that the C_k -PRECOLORING EXTENSION problem can be solved in polynomial time in P_9 -free graphs.

► **Theorem 1.1.** *Let $k \geq 5$ be odd, G be a P_9 -free graph of order n , W be a subset of its vertices, and h be a mapping from W to $V(C_k)$. Then one can determine in $O(n^{12k+3})$ time if h can be extended to a C_k -coloring of G , and find such a C_k -coloring if one exists.*

The algorithm is described in detail in Section 3. It builds on the recent work on 3-COLORING P_7 -free graphs [4]. The high-level idea of the algorithm is the following: First, we partition the graph into a so-called *layer structure* and guess the colors of a constant number of vertices. This precoloring propagates to other vertices using the layer structure, reducing the lists of possible colors. We keep guessing the colors of other vertices, transforming the input instance into a set of $n^{O(k)}$ subinstances of LIST H -COLORING, such that:

- (i) (G, W, f) is a yes-instance of C_k -PRECOLORING EXTENSION if and only if one of these subinstances is a yes-instance of LIST C_k -COLORING; and
- (ii) each subinstance can be solved in polynomial time by a reduction to 2-SAT.

In Section 4, we study the complexity of variants of H -COLORING in F -free graphs and prove the following theorem.

► **Theorem 1.2.** *Let F be a connected graph. If F is not a subgraph of a subdivided claw, then for every odd $k \geq 5$ the C_k -PRECOLORING EXTENSION problem is NP-complete for F -free graphs.*

We prove the theorem in several steps, analyzing the possible structure of F and trimming the hard cases. Observe that the statement of Theorem 1.2 is similar to the previously mentioned result of Alekseev for INDEPENDENT SET [1]. In most cases, we actually prove hardness for the more restricted C_k -COLORING problem.

Finally, in Section 5, we state some open questions for future research.

2 Preliminaries

Let G be a simple graph. For $X \subseteq V(G)$, we denote by $G|X$ the subgraph induced by X , and denote by $G \setminus X$ the graph $G|(V(G) \setminus X)$. We say that X is *connected* if $G|X$ is connected. For two disjoint subsets $A, B \subset V(G)$, we say that A is *complete* to B if every vertex of A is adjacent to every vertex of B , and that A is *anticomplete* to B if every

vertex of A is nonadjacent to every vertex of B . If $A = \{a\}$ we write a is complete (or anticomplete) to B to mean that $\{a\}$ is complete (or anticomplete) to B . For $X \subseteq V(G)$, we say that $e \in E(G)$ is *an edge of X* if both endpoints of e are in X . For $v \in V(G)$ we write $N_G(v)$ (or $N(v)$ when there is no danger of confusion) to mean the set of vertices of G that are adjacent to v . Observe that since G is simple, $v \notin N(v)$. For $X \subseteq V(G)$ we define $N(X) = (\bigcup_{v \in X} N(v)) \setminus X$. We say that the set S *dominates X* , or S is a *dominating set* of X if $X \subseteq S \cup N(S)$. We write that S dominates G when we mean that it dominates $V(G)$. A component of G is *trivial* if it has only one vertex and *nontrivial* otherwise.

We use $[k]$ to denote the set $\{1, 2, \dots, k\}$. We denote by P_t the path with t vertices. A *path in a graph G* is a sequence $v_1 - \dots - v_t$ of pairwise distinct vertices such that for any $i, j \in [t]$, $v_i v_j \in E(G)$ if and only if $|i - j| = 1$. The *length* of this path is t . We denote by $V(P)$ the set $\{v_1, \dots, v_t\}$. If $a, b \in V(P)$, say $a = v_i$ and $b = v_j$ with $i < j$, then $a - P - b$ is the path $v_i - v_{i+1} - \dots - v_j$, and $b - P - a$ is the path $v_j - v_{j-1} - \dots - v_i$.

Let $k \geq 3$ be an odd integer. We denote by C_k a cycle with k vertices $1, 2, \dots, k$ that appear along the cycle in this order. The calculations on vertices of C_k will be preformed modulo k , with 0 unified with vertex k .

We say that (G, L') is a *subinstance* of (G, L) if $L'(v) \subseteq L(v)$ for every $v \in V(G)$. Two C_k -list assignments L and L' of G are *equivalent* if (G, L) is C_k -colorable if and only if (G, L') is C_k -colorable. A C_k -list assignment L is *equivalent* to a set \mathcal{L} of C_k -list assignments of a graph G if there is $L' \in \mathcal{L}$ such that (G, L) is equivalent to (G, L') .

Let (G, L) be an instance of LIST C_k -COLORING. We say that the list $L(x)$ of a vertex x is *good* if $|L(x)| \in \{1, 2, 3, k\}$ and in addition

- if $|L(x)| = 2$, then $L(x) = \{i - 1, i + 1\}$ for some $i \in [k]$, and
- if $|L(x)| = 3$, then $L(x) = \{i, i - 2, i + 2\}$ for some $i \in [k]$.

We say that L is *good* if $L(v)$ is good for all $v \in V(G)$.

For an edge $vw \in E(G)$, we *update v from w* if one of the following is performed.

- If $L(w) = \{i\}$ for some $i \in [k]$, then replace the list of v by $\{i - 1, i + 1\} \cap L(v)$.
- If $L(w) = \{i - 1, i + 1\}$ for some $i \in [k]$, then replace the list of v by $\{i, i + 2, i - 2\} \cap L(v)$.
- If $L(w) = \{i, i - 2, i + 2\}$, $L(v) = \{j, j + 2, j - 2\}$ for some $i, j \in [k]$, then replace the list of v by $\{i - 1, i + 1, i - 3, i + 3\} \cap L(v)$.

Clearly, any update creates an equivalent subinstance of (G, L) . Note that in the graph homomorphism literature this operation is usually referred to as *edge (or arc) consistency* and it is performed in the beginning of most algorithms solving variants of H -COLORING [22, 30]. However, we keep the name “update” to emphasize that we will only perform it at certain points in our algorithm. We say that an update of v from w is *effective* if the size of the list of v decreases by at least 1, and *ineffective* otherwise. Note that an update is effective if and only if there exists an element $c \in L(v)$ which is not an element of $\{i - 1, i + 1\}$, $\{i, i + 2, i - 2\}$ or $\{i - 1, i + 1, i - 3, i + 3\}$ depending on the case in the definition of an update. We observe that the update does not change the goodness of the list².

► **Lemma 2.1** (♠). *If the lists of v and w are good before updating v from w , then the list of v is good or empty after the update.*

A C_k -list assignment L is said to be *reduced* if no effective update can be performed. It is well-known that one can obtain a reduced list assignment in polynomial time.

² The proofs of theorems and lemmas marked with ♠ are omitted due to space constraints.

► **Lemma 2.2** (♠). *Let G be a graph of order n , and L be a C_k -list assignment. There exists an $O(n^3)$ -time algorithm to obtain an equivalent reduced subinstance (G, L') of (G, L) or determine that (G, L) has no C_k -coloring.*

We now introduce two more tools that are important for our purpose. The first one is purely graph-theoretic and describes the structure of P_t -free graphs.

► **Theorem 2.3** ([6]). *Let G be a connected P_t -free graph with $t \geq 4$. Then G has a connected dominating set D such that $G|D$ is either P_{t-2} -free or isomorphic to P_{t-2} .*

The next observation generalizes the observation by Edwards [10] that LIST k -COLORING can be solved in polynomial time, whenever the size of each list is at most two. This was already noted by e.g. Feder and Hell [12].

► **Theorem 2.4** (♠). *Let (G, L) be an instance of LIST H -COLORING where G is of order n and $|L(v)| \leq 2$ for every $v \in V(G)$. Then one can determine in $O(n^2)$ time if (G, L) is H -colorable and find an H -coloring if one exists.*

3 Polynomial algorithm for P_9 -free graphs

In this section, we show that C_k -PRECOLORING EXTENSION can be solved in polynomial time for P_9 -free graphs.

► **Theorem 1.1.** *Let $k \geq 5$ be odd, G be a P_9 -free graph of order n , W be a subset of its vertices, and h be a mapping from W to $V(C_k)$. Then one can determine in $O(n^{12k+3})$ time if h can be extended to a C_k -coloring of G , and find such a C_k -coloring if one exists.*

Outline of the proof

The overall strategy is to reduce the instance (G, W, h) , in polynomial time, to a set \mathcal{I} of polynomially many instances of LIST C_k -COLORING, in which every list has size at most 2, and (G, W, h) is a yes-instance if and only if at least one instance from \mathcal{I} is a yes-instance. We then apply Theorem 2.4 to solve each instance from \mathcal{I} in polynomial time.

More specifically, our algorithm, at a high level, consists of five phases. In the first three of them, we focus on processing the graph $G' := G \setminus W$. First, we apply Theorem 2.3 to show that the vertex set of G' can be partitioned into four sets (S, X, Y, Z) such S is connected and dominates X , X dominates Y , and Y dominates Z . Second, we branch on every possible C_k -coloring of $G'|S$. For each of these colorings of $G'|S$, we propagate the coloring of S to the vertices of $G' \setminus S$ via updates. After updating, the vertices in $S \cup X$ will have lists of size at most 2, but the vertices in $Y \cup Z$ may still have larger lists. In the third phase, we reduce the instance to polynomially many subinstances via branching in such a way that each of the subinstances avoids certain configurations, which we call *bad paths*. Finally, using the fact that each subinstance has no bad paths, in the last two phases we reduce the list size of vertices in $Y \cup Z$ to at most 2, restore the set W , creating a set of instances, which is equivalent to (G, W, h) , and use Theorem 2.4 to solve the created instances in polynomial time.

Proof of Theorem 1.1. We view (G, W, h) as an equivalent instance (G, L) of LIST C_k -COLORING where $L(v) = \{h(v)\}$ if $v \in W$ and $L(v) = [k]$ otherwise.

Clearly, h can be extended to an C_k -coloring of G if and only if (G, L) is C_k -colorable. Moreover, observe that if G contains a triangle, then we can immediately report a no-instance. Checking for existence of triangles can clearly be done in $O(n^3)$ time, so from now on we assume that G is triangle-free.

For the first three phases we consider the graph $G' := G \setminus W$. We may assume that G' is connected, for otherwise we can apply the same reasoning to every connected component.

Phase I. Obtaining a layer structure

Let us start with imposing some structure on the vertices of G' .

▷ **Claim 3.1.** There exists $S \subseteq V(G')$ such that $|S| \leq 7$, the graph $G'|S$ is connected, and $S \cup N(S) \cup N(N(S))$ dominates G' .

Proof. We apply Theorem 2.3 to G' : G' has a connected dominating set D that induces a subgraph that is either P_7 -free or isomorphic to a P_7 . If $G'|D$ is isomorphic to a P_7 , then D is the desired set S . Otherwise we apply Theorem 2.3 on $G'|D$ to conclude that $G'|D$ has a connected dominating set D' that induces a subgraph that is either P_5 -free or isomorphic to a P_5 . If $G'|D'$ is isomorphic to a P_5 , then D' is the desired set S . Otherwise $G'|D'$ is P_5 -free. We again apply Theorem 2.3 on $G'|D'$: $G'|D'$ has a connected dominating set D'' that induces a subgraph that is either P_3 -free or isomorphic to P_3 . Then $D'' \cup N(D'') \cup N(N(D''))$ dominates G' . Since G' is triangle-free, if $G'|D''$ is P_3 -free, then D'' is a clique of size at most 2. It follows that $|D''| \leq 3$ and thus we can choose D'' for S . ◁

Let S be the set given by Claim 3.1. Define $X = N(S)$, $Y = N(N(S)) \setminus S$ and $Z = V(G') \setminus (X \cup Y \cup Z)$. Then (S, X, Y, Z) is a partition of $V(G')$, where S dominates X , X dominates Y and Y dominates Z , and there is no edge between S and $Y \cup Z$ or between X and Z . Moreover, S is connected. Such a quadruple $\mathcal{P} = (S, X, Y, Z)$ is called a *layer structure* of G' . The set S is called the *seed* of \mathcal{P} .

Phase II. Obtaining a canonical C_k -list assignment via updates

We now branch on every possible C_k -coloring of $G'|S$, there are at most k^7 such colorings since $|S| \leq 7$. Note that k^7 is a constant since k is a fixed number. To prove the theorem, therefore, it suffices to determine whether there is a branch, in which the precoloring of $S \cup W$ can be extended to a C_k -coloring of (G, L) in polynomial time.

In the following, we consider a fixed coloring $f : S \rightarrow [k]$, and we continue with the instance (G', L') of LIST C_k -COLORING, where $L'(v) = \{f(v)\}$ if $v \in S$ and $L'(v) = [k]$ otherwise.

We further partition the sets S , X , and Y as follows. For $1 \leq i \leq k$, we define

$$\begin{aligned} S_i &:= \{s \in S : L(s) = \{i\}\}, \\ X_i &:= \{x \in X \setminus (\bigcup_{j=1}^{i-1} X_j) : N(x) \cap S_i \neq \emptyset\}, \\ Y_i &:= \{y \in Y \setminus (\bigcup_{j=1}^{i-1} Y_j) : N(y) \cap X_i \neq \emptyset\}. \end{aligned}$$

Clearly, (X_1, X_2, \dots, X_k) is a partition of X and (Y_1, Y_2, \dots, Y_k) is a partition of Y .

We now perform the following updates for all $1 \leq i \leq k$ in the following order.

- For every edge sx with $s \in S_i$ and $x \in X_i$, we update x from s .
- For every edge xy with $x \in X_i$ and $y \in Y_i$, we update y from x .

We continue to denote the resulting C_k -list assignment by L' . Then $|L'(s)| = 1$ for every $s \in S$, $L'(x) \subseteq \{i-1, i+1\}$ for every $x \in X_i$ and $L'(y) \subseteq \{i, i-2, i+2\}$ for every $y \in Y_i$. We call such a C_k -list assignment L' *canonical* for $\mathcal{P} = (S, \bigcup_{i=1}^k X_i, \bigcup_{i=1}^k Y_i, Z)$.

▷ **Claim 3.2 (♠).** If X_i is not stable, then (G', L') is not C_k -colorable.

Note that one can determine in $O(n^2)$ time if there exists an X_i that is not stable. If so, we stop and correctly determine that (G', L') is not C_k -colorable by Claim 3.2. Otherwise, we may assume that X_i is stable for all $1 \leq i \leq k$ from now on.

Phase III. Eliminating bad paths via branching ($O(n^{12k})$ branches)

In this phase, we shall reduce the instance (G', L') to an equivalent set of polynomially many subinstances so that every subinstance has no bad paths, which we define now.

An induced path $a - b - c$ is a *bad path* in $\mathcal{P} = (S, X, Y, Z) = (S, \bigcup_{i=1}^k X_i, \bigcup_{i=1}^k Y_i, Z)$ if for some $i \in [k]$ we have $a \in Y_i$, $b, c \in (Y \cup Z) \setminus Y_i$, and $\{b, c\}$ is anticomplete to X_i . We call a the *starter* of $a - b - c$. Let \mathcal{Q}_i be the set of all bad paths with starters in Y_i , clearly $|\mathcal{Q}_i| = O(n^3)$.

A vertex $v \in Y_i$ is of *depth at least ℓ* in \mathcal{P} if for every $x \in N(v) \cap X_i$, there exists an induced path $v - x - P$ of length at least ℓ such that $V(P) \subseteq S$. Observe that every vertex in Y is of depth at least 3 to S (because we may assume that $|S| \geq 2$ and so no vertex in X is complete to S since G is triangle-free), and that the starter of a bad path is of depth at most 7 to S since G' is P_9 -free.

Note that for any C_k -coloring of (G', L') and every $i \in [k]$, either there exists a bad path in \mathcal{Q}_i whose starter is colored with a color in $\{i-2, i+2\}$ or the starters of all bad paths in \mathcal{Q}_i are colored with i . This leads to the following branching scheme, which only updates the lists.

Branching.

■ ($2^k = O(1)$ branches.) For every subset $I \subseteq [k]$, we have a branch B_I intended to find possible colorings such that there exists a bad path in \mathcal{Q}_i whose starter is colored with a color in $\{i-2, i+2\}$ if $i \in I$, and all starters of bad paths in \mathcal{Q}_i are colored with color i if $i \notin I$. Clearly, (G', L') is C_k -colorable if and only if at least one of the B_I is a yes-instance. In the following, we fix a branch B_I .

■ ($O(2^k n^{3k}) = O(n^{3k})$ branches.) We further branch to obtain a set of size $O(n^{3k})$ of subinstances within B_I by guessing, for each $i \in I$, a bad path in \mathcal{Q}_i , and guessing the color of its starter from $\{i-2, i+2\}$. The union over all branches B_I of these subinstances is equivalent to (G', L') .

Specifically, for each element $(a_i - b_i - c_i)_{i \in I}$ in $\prod_{i \in I} \mathcal{Q}_i$, we have one branch where we set $L''(a_i) := L'(a_i) \cap \{i-2, i+2\}$ for every $i \in I$, and we set $L''(a) := L'(a) \cap \{i\}$ for every starter a of a bad path in \mathcal{Q}_i for every $i \notin I$. We denote the resulting C_k -list assignment by L'' . For each such branch and for every element $(q_i)_{i \in I}$ in $\prod_{i \in I} L''(a_i)$, we have one branch where $L''(a_i) = \{q_i\}$ for all $i \in I$. It follows that for all $i \in I$ and $x \in X_i \cap N(a_i)$, the only possible color for x is $(q_i + i)/2$, and so we set $L''(x) = \{(q_i + i)/2\}$. Since $L''(a_i) \subseteq \{i-2, i+2\}$ for all $i \in I$, it follows that there are $2^{|I|} \leq 2^k$ branches. Let us fix one such branch and denote the resulting instance by (G', L'') .

■ ($O(k^{3k}) = O(1)$ branches.) We let I^* be the subset of $[k] \setminus I$ of indices i such that \mathcal{Q}_i contains a bad path. For each $i \in I^*$, we choose a bad path $a_i - b_i - c_i$ in \mathcal{Q}_i such that $|N(a_i) \cap X_i|$ is minimum, where the minimum is taken over all bad paths in \mathcal{Q}_i . Choose a vertex $x_i \in N(a_i) \cap X_i$ for each $i \in I^*$. Define

$$Q := \bigcup_{i \in I} \{b_i, c_i\} \cup \bigcup_{i \in I^*} \{b_i, c_i, x_i\},$$

where for $i \in I$, b_i, c_i are two vertices on the bad path we guessed in the previous bullet. We branch on every possible coloring of Q . Since $|Q| \leq 3k$, the number of branches is at most k^{3k} . In the following, we fix a coloring g of Q and denote the resulting subinstance by (G', L''') , where $L'''(v) = \{g(v)\}$ if $v \in Q$ and $L'''(v) = L''(v)$ otherwise.

Obtaining a new layer structure with a canonical C_k -list assignment. We now deal with (G', L''') . Define

$$A = \bigcup_{i \in I} ((N(a_i) \cap X_i) \cup \{a_i, b_i, c_i\}) \cup \bigcup_{i \in I^*} \{x_i, a_i, b_i, c_i\},$$

and note that in L''' , every vertex in A has a list of size at most 1. We update all vertices of G' from all vertices in A and continue to denote the resulting C_k -list assignment by L''' . We now obtain a new partition $\mathcal{P}' = (S', X', Y', Z')$ of G' as follows.

- Let $S' := S \cup A$.
- For each $1 \leq j \leq k$, let $K_j := \emptyset$. For each vertex $v \in Y \cup Z$, if v has a neighbor in S' , let j be the smallest integer in $[k]$ such that there exists a vertex $s \in N(v) \cap S'$ with $L(s) = \{j\}$, and add v to K_j . For each $1 \leq j \leq k$, let $X'_j = (X_j \cup K_j) \setminus A$. Let $X' := \bigcup_{i=1}^k X'_i$.
- For $1 \leq i \leq k$, let Y'_i be the set of vertices in $V(G') \setminus (S' \cup X' \cup (\bigcup_{j < i} Y'_j))$ that have a neighbor in X'_i . Let $Y' := \bigcup_{i=1}^k Y'_i$.
- Let $Z' := V(G') \setminus (S' \cup X' \cup Y')$.

▷ **Claim 3.3 (♠).** The new partition $\mathcal{P}' := (S', X', Y', Z')$ is a layer structure of G' and L''' is a canonical C_k -list assignment for \mathcal{P}' .

▷ **Claim 3.4 (♠).** For every $i \in [k]$ it holds that

1. $X'_i \setminus X_i \subseteq Y \cup Z$.
2. If a vertex in $Y' \cup Z'$ is anticomplete to X'_i , then it is anticomplete to X_i .
3. $Y'_i \setminus Y_i$ is anticomplete to X_i .

The following claim is the key to our branching algorithm.

▷ **Claim 3.5.** Let a be a starter of a bad path in \mathcal{P}' . If the depth of the starter of any bad path in \mathcal{P} is at least ℓ , then the depth of a in \mathcal{P}' is at least $\ell + 1$.

Proof. Let $a' - b' - c'$ be a bad path in \mathcal{P}' with $a' \in Y'_i$. Consider the following cases.

Case 1: $a' \in Y_i \cap Y'_i$. Then $\emptyset \neq N(a') \cap X_i \subseteq X'_i$. By item 2. in Claim 3.4, $\{b', c'\}$ is anticomplete to X_i and so $a' - b' - c'$ is also a bad path in $\mathcal{P} = (S, X, Y, Z)$. This implies that $\mathcal{Q}_i \neq \emptyset$. Therefore, there exist $a, b, c, x \in S'$ such that $a - b - c$ is a bad path in \mathcal{P} with $a \in Y_i$ and $x \in N(a) \cap X_i$.

We first claim that it is possible to pick a vertex $x' \in N(a') \cap X_i$ that is not adjacent to a . Recall that the branch we consider corresponds to a set $I \subseteq [k]$. If $i \in I$, then all vertices in $N(a) \cap X_i$ are in A and hence are now in S' . So every vertex in $N(a') \cap X_i$ is not adjacent to a , and our claim holds. If $i \notin I$, then $i \in I^*$, and so $a = a_i$. By the choice of a_i , it follows that $|N(a) \cap X_i| \leq |N(a') \cap X_i|$. Since $a' \in Y'_i$, it follows that a' is not adjacent to x . Therefore, there exists a vertex $x' \in N(a') \cap X_i$ such that x' is not adjacent to a .

Note that x and x' are not adjacent by Claim 3.2. Moreover, x' is anticomplete to $\{b', c', b, c\}$ by the definition of bad path. Let P' be the shortest path from x to x' with internal vertices contained in S . Note that P' exists since S is connected. Then P' is an

31:10 Complexity of C_k -Coloring in Hereditary Classes of Graphs

induced path. Since $V(P') \setminus \{x, x'\} \subseteq S$, it follows that $V(P') \setminus \{x, x'\}$ is anticomplete to $\{a, b, c, a', b', c'\}$. Therefore, $c - b - a - x - P' - x' - a' - b' - c'$ is an induced path of order at least 9, a contradiction.

Case 2: $a' \in Y'_i \setminus Y_i$. It follows from Claim 3.4, item 3. that $N(a') \cap X'_i \subseteq X'_i \setminus X_i$. Pick a vertex $x' \in N(a') \cap X'_i$. Since $x \in X'_i \setminus X_i$, x' has a neighbor $s' \in S'$ by the definition of X'_i . By Claim 3.4, item 1., $x' \in Y \cup Z$ and so $s' \in S' \setminus S = A$. Thus there exists $j \in I$ such that x' is not anticomplete to $Q = \{x_j, a_j, b_j, c_j\}$, where $x_j \in N(a_j) \cap X_j$. Let $a_j - x_j - P$ be an induced path of length ℓ with $V(P) \subseteq S$. Note that $x' \in Y \cup Z$ is anticomplete to $V(P) \subseteq S$. Let $x' - P'' - x_j$ be the shortest path from x' to x_j such that $V(P'') \subseteq Q$. Since a' is anticomplete to $\{x\} \cup V(P) \cup V(P'') \subseteq S'$, it follows that $a' - x' - P'' - x_j - P$ is an induced path of length at least $\ell + 1$. This proves the claim. \triangleleft

Therefore, we have obtained an equivalent set of subinstances of size $O(n^{3k})$. For each such subinstance, the minimum depth of the starter of a bad path has increased by at least 1 compared to \mathcal{P} due to Claim 3.5. Note that the depth of any starter of a bad path in \mathcal{P} is at least 3. Moreover, since G' is P_9 -free, the depth of any starter of a bad path is at most 7.

By branching 4 times, therefore, we obtain an equivalent set of $O(n^{12k})$ subinstances such that each subinstance has no bad paths.

Phase IV. Reducing the list size of vertices in Z

Now we go back to processing the graph G . Let us fix an instance of LIST C_k -COLORING on G' , created in the previous phase, and let (G, L) denote the instance obtained by restoring the vertices of W . By $\mathcal{P} = (S, X, Y, Z)$ we denote the layer structure of G' with no bad paths and L is canonical for \mathcal{P} . We first reduce the list size of vertices in Z .

\triangleright **Claim 3.6** (\spadesuit). The set Z is stable and each $z \in Z$ has neighbors in at most one of $\{Y_1, Y_2, \dots, Y_k\}$.

\triangleright **Claim 3.7** (\spadesuit). Let $z \in Z$ be anticomplete to W and have a neighbor in Y_i . If (G, L) has a C_k -coloring, then (G, L) has a C_k -coloring c such that $c(z) \in \{i - 1, i + 1\}$.

We now modify the lists of vertices in Z : let $L(z) := L(z) \cap \{i - 1, i + 1\}$ for every $z \in Z$ that is anticomplete to W and has a neighbor in Y_i . It follows from Claim 3.7 that the resulting list is equivalent to the original one. We still denote by the resulting list L .

Phase V. Reducing the list size of vertices in Y

We now apply Lemma 2.2 to obtain a reduced C_k -list assignment L' . Then (G, L') is an equivalent subinstance of (G, L) . If $L'(v) = \emptyset$ for some $v \in V(G)$, we stop and report a no-instance. Define:

$$\begin{aligned} S' &:= \{v \in V(G') : |L'(v)| = 1\}, \\ X'_i &:= \{v \in V(G') \setminus S' : L'(v) \subseteq \{i - 1, i + 1\}\}, 1 \leq i \leq k, \\ Y'_i &:= \{v \in V(G') \setminus (S' \cup X' \cup \bigcup_{j < i} Y'_j) : L'(v) \subseteq \{i, i - 2, i + 2\}\}, 1 \leq i \leq k, \\ X' &:= \bigcup_{i=1}^k X'_i, \\ Y' &:= \bigcup_{i=1}^k Y'_i. \end{aligned}$$

Note $W \subseteq S'$ and that (S', X', Y', \emptyset) is almost a layer structure of G except that S' is not necessarily connected. It follows from the definition of (S', X', Y') and Lemma 2.1 that $L'(x) = \{i-1, i+1\}$ for every $x \in X'_i$ and $L'(y) = \{i-2, i, i+2\}$ for every $y \in Y'_i$. For $1 \leq i \leq k$, we partition Y'_i into two subsets Y_i^1 and Y_i^2 , where Y_i^2 is the set of isolated vertices in $G|Y'_i$ and $Y_i^1 = Y'_i \setminus Y_i^2$. We prove a few properties for S' , X' and Y' .

▷ **Claim 3.8** (♠). The following hold for S' , X' and Y' .

1. $V(G) = S' \cup X' \cup Y'$.
2. For every $i \in [k]$ and $y \in Y'_i$, we have $N(y) \cap X' \subseteq X'_i$.
3. For every $i \in [k]$, we have $X_i \subseteq X'_i \cup S'$ and $Y'_i \subseteq Y_i$.
4. For every $i \in [k]$ and $y \in Y_i^2$, the vertex y is anticomplete to $Y' \setminus (Y_{i+1}^2 \cup Y_{i-1}^2)$.

▷ **Claim 3.9.** If (G, L') is C_k -colorable, then there exists a C_k -coloring c of (G, L') such that for each $1 \leq i \leq k$, $c(y) = i$ for all $y \in Y_i^2$ and $c(y) \in \{i-2, i+2\}$ for all $y \in Y_i^1$.

Proof. Suppose that c' is a C_k -coloring of (G, L') . Note that each $u \in Y_i^1$ has a neighbor $v \in Y_i^1$ by the definition. Since $L'(u), L'(v) \subseteq \{i-2, i, i+2\}$ and c' is a C_k -coloring of (G, L') , it follows that $c'(u) \neq i$ and $c'(v) \neq i$. So $c'(u) \in \{i-2, i+2\}$.

Let $u \in Y_i^2$. Note that u can only have neighbors in X'_i or in $Y_{i+1}^2 \cup Y_{i-1}^2$ by item 2. and item 4. of Claim 3.8. Define $c : V(G) \rightarrow [k]$ such that $c(v) := i$ if $v \in Y_i^2$ and $c(v) := c'(v)$ if $v \notin \bigcup_{i=1}^k Y_i^2$.

Then c is a C_k -coloring of (G, L') , since $c'(x) \in \{i-1, i+1\}$ for every $x \in X'_i$. This completes the proof. ◁

Let us point out that the special treatment of the sets Y_i^1 is needed only for the case $k = 5$. For $k > 5$, if one Y_i contains two adjacent vertices, one can observe that there is no way to color them. Thus we can immediately report a no-instance (or let it be reported when we solve the corresponding 2-SAT instance).

Let us now modify the lists as follows. For each $1 \leq i \leq k$ and each $y \in Y'_i$, let $L'(y) := L'(y) \cap \{i-2, i+2\}$ if $y \in Y_i^1$ and $L'(y) := L'(y) \cap \{i\}$ if $y \in Y_i^2$. By Claim 3.9, the new list assignment is equivalent to the original one. Now for each $v \in V(G)$ we have $|L'(v)| \leq 2$ and so Theorem 2.4 applies.

This completes the proof of correctness of our algorithm. Clearly, the most expensive part of our algorithm is Phase III where we branch into $O(n^{12k})$ subinstances. Since each subinstance can be constructed in $O(n^3)$ time by Lemma 2.2 and each 2-SAT instance can be solved in $O(n^2)$ time by Theorem 2.4, the total running time is $O(n^{12k+3})$. ◀

4 Hardness results

In this section we prove the following theorem.

▶ **Theorem 1.2.** *Let F be a connected graph. If F is not a subgraph of a subdivided claw, then for every odd $k \geq 5$ the C_k -PRECOLORING EXTENSION problem is NP-complete for F -free graphs.*

We will prove Theorem 1.2 in several steps in which we analyze possible structure of F . We start with the following simple observation that will be repeatedly used. For the rest of this section, let $k = 2s + 1$ for $s \geq 2$.

▶ **Observation 4.1.** *Let $s \geq 2$ be an integer and P be a $2s$ -vertex path with endvertices a and b . Then the following holds.*

- *In any C_{2s+1} -coloring h of P we have $h(a) \neq h(b)$.*
- *For any distinct $i, j \in \{1, 2, \dots, 2s+1\}$, there exists a C_{2s+1} -coloring h of P such that $h(a) = i$ and $h(b) = j$.*

4.1 Eliminate cycles

The *girth* of a graph G , denoted by $\text{girth}(G)$, is the length of a shortest cycle in G . A vertex in a graph is called a *branch vertex* if its degree is at least 3. By Γ_p we denote the class of graphs, in which the number of edges in any path joining two branch vertices is divisible by p .

We first show that the problem is NP-hard in F -free graphs, unless F is a tree in Γ_{2s-1} .

► **Theorem 4.2.** *For each fixed integer $s \geq 2$ and each connected graph F , C_{2s+1} -COLORING is NP-complete for F -free graphs whenever F contains a cycle or is not in Γ_{2s-1} .*

Proof. It is known (see e.g. [31]) that the $(2s+1)$ -COLORING problem is NP-complete for graphs of girth at least g for each fixed $g \geq 3$. We reduce this problem to C_{2s+1} -COLORING. Given a graph G , we obtain a graph G' by replacing each edge of G by a $(2s-1)$ -edge path. Then it follows from Observation 4.1 that G is $(2s+1)$ -colorable if and only if G' is C_{2s+1} -colorable. Clearly, $\text{girth}(G') = \text{girth}(G) \cdot (2s-1) \geq g(2s-1)$. Thus, if we choose $g \geq 3$ such that $g(2s-1) > \text{girth}(F)$, e.g., $g = |V(F)| + 1$, it follows that all graphs of girth at least $g(2s-1)$ are F -free. Moreover, it is easy to see that the number of edges in any path joining two branch vertices of G' is divisible by $2s-1$, so if $F \notin \Gamma_{2s-1}$, then G' does not contain F . ◀

4.2 Eliminate vertices of degree at least 4

From now on it suffices to consider trees with branch vertices at distance divisible by $2s-1$. We now show that C_k -COLORING is NP-complete for F -free graphs if F contains a vertex of degree at least 4. Note that in this case every subcubic graph is F -free.

► **Theorem 4.3 (♠).** *For each fixed $s \geq 2$, C_{2s+1} -COLORING is NP-complete for subcubic graphs.*

4.3 Eliminate multiple branch vertices

Before we prove the main theorem we need one more intermediate step that allows us to eliminate those F in which there are two branch vertices that are at distance not divisible by s . The proof is a reduction from the problem called NON-RAINBOW COLORING EXTENSION, whose instance is a 3-uniform hypergraph H and a partial coloring f of some of its vertices with colors $\{1, 2, 3\}$. We ask whether f can be extended to a 3-coloring of $V(H)$ such that no hyperedge is *rainbow* (i.e., contains three distinct colors). This problem is known to be NP-complete [3].

► **Theorem 4.4.** *For each fixed integer $s \geq 2$, C_{2s+1} -PRECOLORING EXTENSION is NP-complete for bipartite graphs in Γ_s .*

Proof. We reduce from NON-RAINBOW COLORING EXTENSION. Let $H = (V, E)$ be a 3-uniform hypergraph and let f be a partial 3-coloring of H . We construct an instance of C_{2s+1} -PRECOLORING EXTENSION as follows.

- For each vertex $v \in V$, we introduce a variable vertex, denoted by v' . If v is precolored by f , we precolor v' with the color $f(v)$.
- For each v that is not precolored by f , we introduce $2s-2$ new vertices and precolor them with $4, 5, \dots, 2s+1$, respectively. Then each of these new vertices is joined by a $(2s-1)$ -edge path to v' . It follows from Observation 4.1 that each vertex v' can only be mapped to one of $1, 2, 3$, and any of these three choices is possible.
- For each hyperedge $e = \{x, y, z\} \in E$, we add a new vertex v_e and three s -edge paths connecting v_e to x', y' , and z' , respectively. This whole subgraph is called an *edge gadget*.

Observe that if x' is mapped to $i \in \{1, 2, 3\}$, then the possible colors for v_e are $\{s+i, s+i-2, \dots, s+i-2\lfloor s/2 \rfloor\} \cup \{s+i+1, s+i+3, \dots, s+i+1+2\lfloor s/2 \rfloor\}$. Thus, if each of x', y', z' is mapped to a different vertex from $\{1, 2, 3\}$, then there is no way to extend this mapping to the whole edge gadget. On the other hand, such an extension is possible whenever x', y', z' receive at most two distinct colors.

We denote by G the resulting graph. By the properties of variable vertices and edge gadgets, (H, f) is a yes-instance of NON-RAINBOW COLORING EXTENSION if and only if the precoloring of G can be extended to a C_{2s+1} -coloring of G . Clearly, G is bipartite and belongs to Γ_s . ◀

By Theorems 4.2, 4.3, and 4.4, the C_{2s+1} -PRECOLORING EXTENSION problem is NP-complete for F -free graphs unless F is a tree in $\Gamma_{s(2s-1)}$ (observe that s and $2s-1$ are relatively prime). We are now ready to show that the problem is NP-hard if F has more than one branch vertex.

► **Theorem 4.5.** *Let $s \geq 2$ be an integer and let F be a tree. If F contains two branch vertices, then C_{2s+1} -COLORING is NP-complete for F -free graphs.*

Proof. Let d be the distance between two closest branch vertices in F . We reduce from POSITIVE NOT-ALL-EQUAL SAT with all clauses containing exactly three literals. Consider an instance with variables x_1, x_2, \dots, x_n and clauses D_1, D_2, \dots, D_m .

- We start our construction by introducing one special vertex z .
- For each variable x_i , we introduce a vertex v_i , adjacent to z .
- For each clause $D_\ell = \{x_i, x_j, x_k\}$, we introduce three new vertices $y_{\ell,i}, y_{\ell,j}$, and $y_{\ell,k}$, and join each pair of them with a $(2s-1)$ -edge path. This guarantees that in every C_{2s+1} -coloring, they get three distinct colors. These three paths constitute the *clause gadget*.
- For each variable x_i belonging to a clause D_ℓ , we join each $y_{\ell,i}$ to v_i by a path $P_{\ell,i}$ with $2d(2s-1) + 1$ edges. Let $v_i = p_1, p_2, \dots, p_{2d(2s-1)+2} = y_{\ell,i}$ be the consecutive vertices of $P_{\ell,i}$. We add edges joining z and $p_{1+j(2s-1)}$ for every $1 \leq j \leq 2d$.

This completes the construction of a graph G . We claim that G is C_{2s+1} -colorable if and only if the initial formula is satisfiable, and that G belongs to our class.

▷ Claim 4.6 (♠). G is C_{2s+1} -colorable if and only if the initial formula is satisfiable.

▷ Claim 4.7 (♠). G is F -free.

This completes the proof of Theorem 4.5. ◀

Now Theorem 1.2 comes from combining the Theorems 4.2, 4.3, 4.4, and 4.5. We observe that all reductions in our hardness proofs are linear in the number of vertices (the target graph is assumed to be fixed, so s is a constant). Moreover, all problems we are reducing from can be shown to be NP-complete by a linear reduction from 3-SAT. Thus we get the following result, conditioned on the Exponential Time Hypothesis (ETH), which, along with the *sparsification lemma*, implies that 3-SAT with n variables and n clauses cannot be solved in time $2^{o(n+m)}$ [27, 28].

► **Corollary 4.8.** *Unless the ETH fails, the following holds. If F is a connected graph that is not a subgraph of a subdivided claw, then for every $s \geq 2$, the C_{2s+1} -PRECOLORING EXTENSION problem cannot be solved in time $2^{o(n)}$ in F -free graphs with n vertices.*

5 Conclusion

In this paper, we initiate a study of C_{2s+1} -COLORING for F -free graphs for a fixed graph F . We prove that C_{2s+1} -PRECOLORING EXTENSION is NP-complete for F -free graphs if some component of F is not a subdivided claw. Moreover, we show that C_{2s+1} -PRECOLORING EXTENSION is polynomial-time solvable for P_9 -free graphs. Note that all our hardness results work for C_{2s+1} -COLORING, except for Theorem 4.4. Thus it is natural to ask whether analogous hardness results holds for C_{2s+1} -COLORING too. Moreover, the following questions seem natural to explore.

- Are there values of s and t such that C_{2s+1} -COLORING is NP-complete for P_t -free graphs?
- Is C_{2s+1} -COLORING polynomial for F -free graphs when F is a subdivided claw?
- Is C_{2s+1} -COLORING FPT for P_t -free graphs, when parameterized by s ?

References

- 1 Vladimir E Alekseev. The effect of local constraints on the complexity of determination of the graph independence number. *Combinatorial-algebraic methods in applied mathematics*, pages 3–13, 1982.
- 2 Gábor Bacsó, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Zolt Tuza, and Erik Jan van Leeuwen. Subexponential-Time Algorithms for Maximum Independent Set in P_t -Free and Broom-Free Graphs. *Algorithmica*, 81(2):421–438, 2019. doi:10.1007/s00453-018-0479-5.
- 3 Manuel Bodirsky, Jan Kára, and Barnaby Martin. The complexity of surjective homomorphism problems – a survey. *Discrete Applied Mathematics*, 160(12):1680–1690, 2012. doi:10.1016/j.dam.2012.03.029.
- 4 Flavia Bonomo, Maria Chudnovsky, Peter Maceli, Oliver Schaudt, Maya Stein, and Mingxian Zhong. Three-Coloring and List Three-Coloring of Graphs Without Induced Paths on Seven Vertices. *Combinatorica*, 38(4):779–801, 2018. doi:10.1007/s00493-017-3553-8.
- 5 Andrei A. Bulatov. A Dichotomy Theorem for Nonuniform CSPs. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 319–330. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.37.
- 6 Eglantine Camby and Oliver Schaudt. A New Characterization of P_k -Free Graphs. *Algorithmica*, 75(1):205–217, 2016. doi:10.1007/s00453-015-9989-6.
- 7 Maria Chudnovsky, Sophie Spirkl, and Mingxian Zhong. Four-coloring P_6 -free graphs. I. Extending an excellent precoloring. *CoRR*, abs/1802.02282, 2018. arXiv:1802.02282.
- 8 Maria Chudnovsky, Sophie Spirkl, and Mingxian Zhong. Four-coloring P_6 -free graphs. II. Finding an excellent precoloring. *CoRR*, abs/1802.02283, 2018. arXiv:1802.02283.
- 9 Maria Chudnovsky, Sophie Spirkl, and Mingxian Zhong. Four-coloring P_6 -free graphs. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1239–1256. SIAM, 2019. doi:10.1137/1.9781611975482.76.
- 10 Keith Edwards. The Complexity of Colouring Problems on Dense Graphs. *Theor. Comput. Sci.*, 43:337–343, 1986. doi:10.1016/0304-3975(86)90184-2.
- 11 Jessica Enright, Lorna Stewart, and Gábor Tardos. On List Coloring and List Homomorphism of Permutation and Interval Graphs. *SIAM J. Discrete Math.*, 28(4):1675–1685, 2014. doi:10.1137/13090465X.
- 12 Tomas Feder and Pavol Hell. List Homomorphisms to Reflexive Graphs. *J. Comb. Theory, Ser. B*, 72(2):236–250, 1998. doi:10.1006/jctb.1997.1812.
- 13 Tomás Feder, Pavol Hell, and Jing Huang. List Homomorphisms and Circular Arc Graphs. *Combinatorica*, 19(4):487–505, 1999. doi:10.1007/s004939970003.
- 14 Tomás Feder, Pavol Hell, Sulamita Klein, Loana Tito Nogueira, and Fábio Protti. List matrix partitions of chordal graphs. *Theor. Comput. Sci.*, 349(1):52–66, 2005. doi:10.1016/j.tcs.2005.09.030.

- 15 Tomás Feder and Moshe Y. Vardi. Monotone monadic SNP and constraint satisfaction. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 612–622. ACM, 1993. doi:10.1145/167088.167245.
- 16 Anna Galluccio, Pavol Hell, and Jaroslav Nesetril. The complexity of H -colouring of bounded degree graphs. *Discrete Mathematics*, 222(1-3):101–109, 2000. doi:10.1016/S0012-365X(00)00009-1.
- 17 Petr A. Golovach, Matthew Johnson, Daniël Paulusma, and Jian Song. A Survey on the Computational Complexity of Coloring Graphs with Forbidden Subgraphs. *Journal of Graph Theory*, 84(4):331–363, 2017. doi:10.1002/jgt.22028.
- 18 Petr A. Golovach, Daniël Paulusma, and Jian Song. Closing complexity gaps for coloring problems on H -free graphs. *Inf. Comput.*, 237:204–214, 2014. doi:10.1016/j.ic.2014.02.004.
- 19 Carla Groenland, Karolina Okrasa, Paweł Rzażewski, Alex Scott, Paul Seymour, and Sophie Spirkl. H -colouring P_t -free graphs in subexponential time. *Discrete Applied Mathematics*, (to appear), 2019.
- 20 Andrzej Grzesik, Tereza Klimošová, Marcin Pilipczuk, and Michal Pilipczuk. Polynomial-time algorithm for Maximum Weight Independent Set on P_6 -free graphs. *CoRR*, abs/1707.05491, 2017. arXiv:1707.05491.
- 21 Andrzej Grzesik, Tereza Klimošová, Marcin Pilipczuk, and Michal Pilipczuk. Polynomial-time algorithm for Maximum Weight Independent Set on P_6 -free graphs. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1257–1271. SIAM, 2019. doi:10.1137/1.9781611975482.77.
- 22 Pavol Hell and Jaroslav Nesetril. *Graphs and Homomorphisms*. Oxford University Press, July 2004. doi:10.1093/acprof:oso/9780198528173.001.0001.
- 23 Pavol Hell and Jaroslav Nešetřil. On the complexity of H -coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990. doi:10.1016/0095-8956(90)90132-J.
- 24 Chinh T. Hoàng, Marcin Kamiński, Vadim V. Lozin, Joe Sawada, and Xiao Shu. Deciding k -Colorability of P_5 -Free Graphs in Polynomial Time. *Algorithmica*, 57(1):74–81, 2010. doi:10.1007/s00453-008-9197-8.
- 25 Ian Holyer. The NP-Completeness of Edge-Coloring. *SIAM J. Comput.*, 10(4):718–720, 1981. doi:10.1137/0210055.
- 26 Shenwei Huang. Improved complexity results on k -coloring P_t -free graphs. *Eur. J. Comb.*, 51:336–346, 2016. doi:10.1016/j.ejc.2015.06.005.
- 27 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.
- 28 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.
- 29 Jan Kratochvíl. Precoloring extension with fixed color bound. *Acta Mathematica Universitatis Comenianae. New Series*, 62, January 1993.
- 30 Benoit Larose and Adrien Lemaître. List-homomorphism problems on graphs and arc consistency. *Discrete Mathematics*, 313(22):2525–2537, 2013. doi:10.1016/j.disc.2013.07.018.
- 31 Vadim V. Lozin and Marcin Kamiński. Coloring edges and vertices of graphs without short or long cycles. *Contributions to Discrete Mathematics*, 2(1), 2007. URL: <http://cdm.ucalgary.ca/cdm/index.php/cdm/article/view/60>.
- 32 Gerhard J. Woeginger and Jiri Sgall. The complexity of coloring graphs without long induced paths. *Acta Cybern.*, 15(1):107–117, 2001. URL: http://www.inf.u-szeged.hu/actacybernetica/edb/vol15n1/Woeginger_2001_ActaCybernetica.xml.
- 33 Dmitriy Zhuk. A Proof of CSP Dichotomy Conjecture. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 331–342. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.38.

Consistent Digital Curved Rays and Pseudoline Arrangements

Jinhee Chun

GSIS Tohoku University, Sendai, Miyagi, Japan
jinhee@dais.is.tohoku.ac.jp

Kenya Kikuchi

GSIS Tohoku University, Sendai, Miyagi, Japan
kenya@dais.is.tohoku.ac.jp

Takeshi Tokuyama

Kwansei-Gakuin University, Sanda, Hyōgo, Japan
tokuyama@kwansei.ac.jp

Abstract

Representing a family of geometric objects in the digital world where each object is represented by a set of pixels is a basic problem in graphics and computational geometry. One important criterion is the consistency, where the intersection pattern of the objects should be consistent with axioms of the Euclidean geometry, e.g., the intersection of two lines should be a single connected component. Previously, the set of linear rays and segments has been considered. In this paper, we extended this theory to families of curved rays going through the origin. We further consider some pseudoline arrangements obtained as unions of such families of rays.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Computational Geometry, Digital Geometry, Spanning Tree, Graph Drawing

Digital Object Identifier 10.4230/LIPICs.ESA.2019.32

Funding This work is partially supported by MEXT JSPS Kakenhi 17K19954, 17K00002 and 18H05291.

1 Introduction

The representation of geometric objects in the pixel world does not always satisfy geometric properties such as Euclidean axioms. Figure 1 shows that a naive definition of digital lines may cause inconsistency. In Figure 1, the intersection of a pair of digital lines is divided into three connected components (in the 4-neighbor topology), while it is desired that the intersection should be connected to imitate the Euclidean axiom that two non-parallel lines intersect at a point. Thus, it is important to seek for a digital representation of a family of geometric objects such that they satisfy a digital version of geometric axioms.

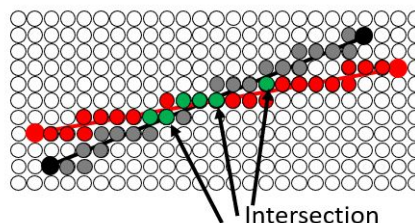


Figure 1 Inconsistency of intersection (green pixels) of two digital line segments.



© Jinhee Chun, Kenya Kikuchi, and Takeshi Tokuyama;

licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 32; pp. 32:1–32:16

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Geometric consistency is important in implementation of algorithms of computational geometry. In geometric computation, we often experience that finite-precision computation is suffered by geometric inconsistency. For example, the divide and conquer algorithm to construct a Voronoi diagram given in the textbook of Preparata and Shamos [6] is known to be difficult to implement. The algorithm needs to compute the intersection point of two (possibly nearly parallel) lines, and then later decides whether the intersection point is above or below another line. Therefore, we may need to compute the intersection point precisely beyond the precision of the system to avoid inconsistency causing a wrong decision. It is a difficult task to avoid such geometric inconsistency. After the seminal paper of Greene and Yao [7], many approaches to overcome the geometric inconsistency in finite precision computation have been proposed. Snap rounding [10, 8, 9] is one of the approaches, which systematically replaces line segments with piecewise linear segments. Another approach implemented in several softwares is the dynamic control of the precision [15, 13].

The pixel-based consistent representation of digital objects would lead to an additional methodology for consistent geometric computation. In general, it is a difficult task to convert families of geometric objects into families of digital objects without geometric inconsistency. However, we have hope if we restrict the task on some fundamental curves to represent basic geometric objects.

In this paper, we propose the *consistent digital curved rays* generalizing consistent digital rays for straight lines [5, 14]. We also show constructions of digital rays that consistently approximate some pseudoline arrangements in the first quadrant.

We consider the triangular region Δ defined by $\{(x, y) : x \geq 0, y \geq 0, x + y \leq n\}$ in the plane, and the integer grid $G = \{(i, j) : i, j \in \{0, 1, \dots, n\}, i + j \leq n\}$ in the region. We can also handle a square region, but use Δ to make the description easier.

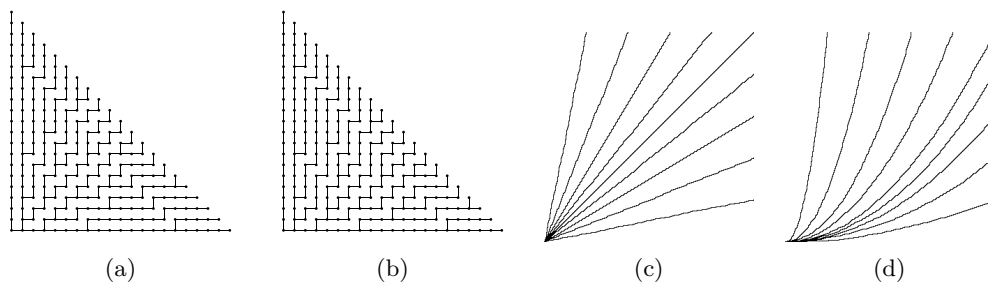
Each element of G is called a *pixel* (usually, a pixel is a square, but we represent it by its lower-left-corner grid point in this paper). A pixel is called a *boundary pixel* if it lies on $x + y = n$. We consider an undirected graph structure under the four-neighbor topology such that $(i, j) \in G$ is connected to $(k, \ell) \in G$ if $(k, \ell) \in \{(i - 1, j), (i, j - 1), (i + 1, j), (i, j + 1)\}$.

A *digital ray* $S(p)$ is a path in G from the origin o to p , where $S(o) = \{o\}$ is a zero-length path. Let us consider a family $\{S(p) : p \in G\}$ of digital rays, where a digital ray is uniquely assigned to each $p \in G$. The family is called *consistent* if the following three conditions hold:

1. If $q \in S(p)$, then $S(q) \subseteq S(p)$.
2. For each $S(p)$, there is a (not necessarily unique) boundary pixel r such that $S(p) \subseteq S(r)$.
3. Each $S(p)$ is a shortest path from o to p in G .

The consistency implies that the union of paths $S(p)$ form a spanning tree T of G such that all leaves are boundary pixels, and accordingly the intersection of two digital rays consists of single connected component. See the pictures (a) and (b) of Figure 2 for the illustration. The tree T and also the family of digital rays are called CDR (*Consistent Digital Rays*).

Previously, the theory has been considered only for digital straightness[11]. Lubby [14] first gave a construction of CDR where each $S(p)$ simulates a linear ray within Hausdorff distance $O(\log n)$, and showed that the bound is asymptotically tight. Here, the Hausdorff distance between objects P and Q is $\max\{\max_{p \in P} \min_{q \in Q} d(p, q), \max_{q \in Q} \min_{p \in P} d(p, q)\}$, where $d(p, q)$ is the Euclidean distance between p and q . The construction was re-discovered by Chun et al.[5] to give further investigation, and Christ et al.[4] gave a construction of consistent digital line segments where the lines need not go through the origin. There are several works on different characterizations and variations [1, 2, 3].



■ **Figure 2** CDR for linear rays and parabolic rays in the triangular region of a 20×20 grid, and sampled linear and parabola digital rays in a 300×300 square grid.

We will extend the theory to families of curves with the same topology as linear rays. In Figure 2, the combinatorial difference between two CDRs can be observed. The difference leads to the visual difference of digital rays illustrated in Figure 2, where it can be seen that the digital rays in (b) approximate parabolas as shown in (d) extended to a sufficiently large grid, while (a) approximates linear rays as shown in (c).

A family \mathcal{F} of nondecreasing curves in Δ is called a *ray family* if each curve goes through the origin o , and for each point $(x, y) \in \Delta \setminus \{o\}$ there exists a unique curve of \mathcal{F} going through it. We call an element of \mathcal{F} a *ray*. Accordingly, each pair of rays intersect each other only at the origin. A typical example is the family of parabolas $y = ax^2$ for $a \geq 0$.

We give a construction method of CDR $T_{\mathcal{F}}$ in G such that the (unique) ray of \mathcal{F} connecting o and a pixel p is approximated by the path $S(p)$ of $T_{\mathcal{F}}$ well. In order to theoretically guarantee the goodness of the approximation, we give an $O(\sqrt{n \log n})$ bound of the Hausdorff distance for several ray families, where the unit is given by the pixel size. Although the theoretical bound is much worse than the known $\Theta(\log n)$ optimal bound for the linear ray [5, 14], it is the first nontrivial result for curved rays as far as the authors know.

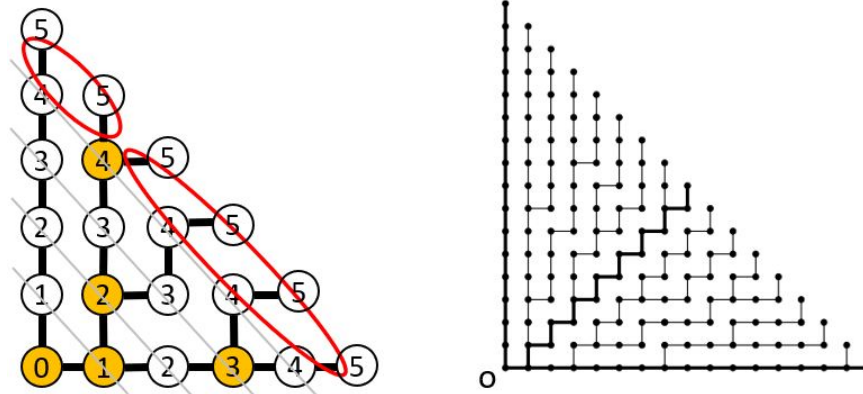
Then, we investigate the structure of unions of CDRs. Our results include a new interpretation of CDS, and generalize it to a digitized pseudoline arrangement (i.e. set of paths intersecting at most once to each other) given as union of translated copies of a ray family. Moreover, we deal with digitization of the arrangement given as a union of families of constant degree homogeneous polynomial curves to show that they can be consistently discretized to form a pseudoline arrangement in a subregion of Δ excluding a constant number of rows and columns, and a constant-area triangle.

We have implemented our construction algorithm of CDR for several families of rays, and our experimental result shows that the Hausdorff distance is only 12 for $n = 2^{14}$ for the parabola rays.

2 Consistent digital rays and their properties

The set of pixels of G on the diagonal $x + y = k$ for $k = 0, 1, \dots, n$ is called the level set $L(k)$. We implicitly give a direction of edges from lower towards higher levels, and call an edge of G between nodes $u \in L(k - 1)$ and $v \in L(k)$ an incoming edge to (resp. outgoing edge from) v (resp. u).

Consider a CDR T . Any node of T has exactly one incoming edge, and at most two outgoing edges of T . The following observation was given by Chun et al.[5] (see Figure 3 for its illustration).



■ **Figure 3** The branching nodes (colored yellow) and partition of incoming edges to the 5th level (left picture) of the CDR (right picture) of linear rays.

► **Lemma 1.** *In the level set $L(k)$ for $k \geq 1$, there exists a real value $0 < x(k) < k$ such that the incoming edge of T to each node whose x -value is smaller than (resp. larger than or equal to) $x(k)$ is vertical (resp. horizontal). Accordingly, there exists a unique branching node of T in $L(k - 1)$ (colored yellow in Figure 3).*

Thus, a CDR is completely characterized by the integer sequence $\lceil x(1) \rceil, \lceil x(2) \rceil, \dots, \lceil x(n) \rceil$, where $1 \leq x(i) \leq i$. We call $x(k)$ the *separating position* on $L(k)$. The following lemma is easy to verify.

► **Lemma 2.** *A (unique) CDR exists for each of $(n - 1)!$ possible sequences as above.*

Our task is to find a CDR among those candidates to approximate a given family of rays as good as possible.

2.1 CDR for linear rays revisited

The CDR of linear rays can be obtained by selecting $x(k)$ as uniformly as possible from $[1, k]$.

Let us consider the binary representation $k = \sum_{i=0}^{\infty} a(i)2^i$ of a natural number k . The van der Corput sequence (see [12]) is the sequence that is defined by a function $V(k) = \sum_{i=1}^{\infty} a(i)2^{-i}$ from natural numbers to $[0, 1]$. We remove $V(0) = 0$ from our consideration so that the range becomes $(0, 1]$. For example, for $6 = 2 + 4 = 110_2$, $V(6) = 0.11_2 = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}$. Here, a sequence of digits with subscript 2 means 2-adic representation of numbers.

The van der Corput sequence is known to be a low discrepancy sequence: There is a nonnegative constant c such that for each n and a range $[a, b]$ in $(0, 1]$, the number of $k \leq n$ satisfying $V(k) \in [a, b]$ differs from $(b - a)n$ at most $c \log n$. In particular, for each $m < n$, the set $\{V(i) : m \leq i \leq n\}$ gives an almost uniform distribution on $[0, 1]$.

We can set $x(k) = kV(k)$ to obtain a CDR. This CDR is exactly same as the one given by Chun et al.[5], and it has been shown that it approximates the linear rays emanating from the origin with the optimal $\Theta(\log n)$ distance bound. In order to generalize to the curved rays, we give the following interpretation.

Consider a line $y = ax$ intersecting $x + y = k$ at $q = (x_0, k - x_0)$. By definition, its slope is a , which is $\frac{k-x_0}{x_0}$. Naturally, we need to approximate the line segment of slope $\frac{k-x_0}{x_0}$ with a grid path in a neighborhood of q in order to globally approximate a line by the path. Ideally the ratio of vertical edges to the horizontal edges in the path should be $\frac{k-x_0}{x_0}$ in the neighborhood. If we set $x_0 = kt_0$, the ratio is $\frac{1-t_0}{t_0}$.

By the definition of the separating position $x(k)$, the edge incoming to q is vertical if and only if q lies on the left of $x(k)$. Let $x(k) = kt(k)$ and we take $t(k) = \theta$ for a uniformly random variable θ on $(0, 1]$. Then, q is on the left of $x(k)$ if and only if $t_0 < \theta$, and its probability is $1 - t_0$. Thus, the incoming edge becomes horizontal and vertical with probabilities t_0 and $1 - t_0$, respectively. Hence, the ratio between them is $\frac{1-t_0}{t_0}$ as desired.

The construction can be derandomized by replacing θ by $V(k)$ for each k . This derandomization also improves the Hausdorff distance bound.

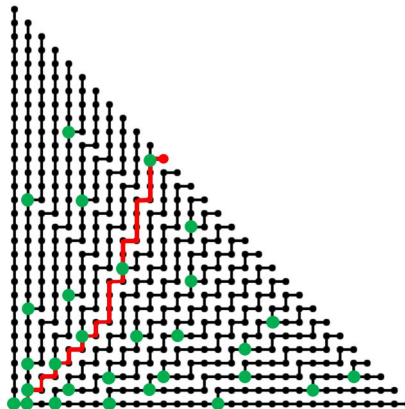
We would like to extend this argument for other families of curves.

3 CDR for families of curves

Let us give a construction method of CDR applicable to several families of curves. We start with a family of parabolas as a typical example for improving the readability, and then discuss more general cases for which we will prove an upper bound for the Hausdorff distance between rays and digital rays.

3.1 CDR for a family of parabolas

3.1.1 Construction of CDR



■ **Figure 4** CDR T_{para} . Green nodes are branching nodes. Red path gives a digital parabolic ray.

Let us consider the family $y = ax^2$ ($a \geq 0$) of parabolas that have the origin o as their apex. We include the y -axis $x = 0$ in the family (this convention is applied to all other cases).

Consider a parabola $C : y = ax^2$ intersecting the level $x + y = k$ at $q = (x_0, k - x_0)$. The slope of tangent at q is $2ax_0$, which is $\frac{2y_0}{x_0} = \frac{2(k-x_0)}{x_0} = \frac{2(1-t_0)}{t_0}$ if we set $x_0 = kt_0$.

Analogously to the linear case, if we would like to have a digital ray nicely approximate C , the curve C in a neighborhood of q should be approximated by a path that contains the horizontal and vertical edges with the probabilities $\frac{t_0}{2-t_0}$ and $\frac{2(1-t_0)}{2-t_0}$, respectively.

Thus, we should select the separating position $x(k) = kt(k)$ to be located on the left of q with probability $\frac{t_0}{2-t_0}$. We consider a monotonically increasing function F_k in the range $[0, 1]$ and set $t(k) = F_k(\theta)$ for the uniformly random variable θ on $(0, 1]$. The probability that $x_0 = kt_0 < x(k)$ is the probability that $F_k^{-1}(t_0) < \theta$ from the monotonicity of F_k . Because of uniformity, this probability equals to $F_k^{-1}(t_0)$.

Thus, we should set $F_k^{-1}(t) = \frac{t}{2-t}$ to meet our requirement, and $F_k(\theta) = \frac{2\theta}{\theta+1}$. This is indeed monotonically increasing as we desired¹. We then derandomize the process by replacing θ with $V(k)$, and we set $t(k) = \frac{2V(k)}{V(k)+1}$ and hence $x(k) = \frac{2kV(k)}{V(k)+1}$ to construct a CDR T_{para} illustrated in Figure 4 deterministically.

The bound for the Hausdorff distance between a parabola ray and its corresponding digital ray in T_{para} is given in the following theorem, although its proof will be given later for a more general form.

► **Theorem 3.** *For each node $p \in G$, the Hausdorff distance between the parabola ray going through p and the path $S(p)$ from p towards the origin in the CDR T_{para} is $O(\sqrt{n \log n})$.*

3.2 Homogeneous polynomials

Let us consider the family \mathcal{F}_j of curves defined by $y = f_a(x) = ax^j$ for $a \geq 0$. Here, the slope of tangent of a curve at (x, y) is $f'_a(x) = jax^{j-1}$, which equals jy/x . Thus, analogously to the parabola case, we have $F_k^{-1}(t) = \frac{t}{j-(j-1)t}$ and $F_k(\theta) = \frac{j\theta}{1+(j-1)\theta}$. Applying derandomization to replace θ by $V(k)$, we set $x(k) = \frac{jkV(k)}{(j-1)V(k)+1}$ for $k = 1, 2, \dots, n$ to define a CDR $T_{\mathcal{F}_j}$. The following theorem is analogously obtained to the parabola case.

► **Theorem 4.** *For each node $p \in G$, the Hausdorff distance between the ray in \mathcal{F}_j going through p and the path $S(p)$ from p towards the origin in the CDR $T_{\mathcal{F}_j}$ is $O(\sqrt{n \log n})$.*

3.3 Handling general ray families

3.3.1 Framework for a diffused ray family

Recall that a family \mathcal{F} of nondecreasing curves in Δ is called *ray family* if each curve (called ray) goes through the origin o , and for each point $(x, y) \in \Delta \setminus \{o\}$ there exists a unique curve of \mathcal{F} going through it. We call a ray family *smooth* if every curve is differentiable. Let us consider the slope $\tau(t, z)$ at $p = (tz, z - tz)$ ($0 < t \leq 1$) of the unique curve of \mathcal{F} going through p . We assume that we can compute $\tau(t, z)$ for a given p efficiently, and its computation time will be regarded as the unit of the time complexity.

► **Definition 5.** *A smooth ray family \mathcal{F} is called *diffused* (resp. *weakly diffused*) if $\tau(t, z)$ is continuous and decreasing (resp. nonincreasing) in t for each fixed $z > 0$.*

Intuitively, the diffusedness means that the rays always expand: The distance between two curves along the off-diagonal $x + y = k$ is increasing in k , since the right curve has a smaller slope than the left one. It can be considered as a continuous counterpart of the property of CDR given in Lemma 1 that vertical edges are incoming to the left of $x(k) = kt(k)$ while horizontal edges incoming to the right of it in each level $L(k)$. The families of parabolas and homogeneous polynomials are diffused.

Now, we consider construction of a CDR for a diffused family \mathcal{F} . We want to control so that the probability that the edge incoming to a pixel $q = (tk, k - tk)$ in $L(k)$ is horizontal with probability $g_k(t) = \frac{1}{1+\tau(t, k)}$, so that the ratio of probabilities to have a vertical edge against a horizontal edge becomes $\tau(t, k)$ for each of $t = j/k$ ($j = 1, 2, \dots, k$).

We would like to find a monotonically increasing function F_k such that $t(k) = F_k(\theta)$ for a uniformly random variable θ on $(0, 1]$ so that the incoming edge to $(tk, k - tk)$ becomes horizontal with probability $g_k(t)$.

¹ The function F_k is independent of k , but it is not always true for the more general cases.

Since the family is diffused, $\tau(t, k)$ is decreasing in t , and hence g_k is increasing. Therefore, it has the inverse function g_k^{-1} that is also increasing.

We set $F_k = g_k^{-1}$ to attain our requirement. Indeed, the condition that $x(k)$ lies on the left of q is that $t(k) < t$, which means $g_k(t(k)) \leq g_k(t)$ because of the monotonicity of g_k . Since $g_k(t(k)) = g_k(F_k(\theta)) = \theta$, this happens if the value of θ is smaller than $g_k(t)$, and hence the probability is $g_k(t)$ as we desire.

By evaluating $F_k(\theta)$ at a given θ , we have $t(k)$ for $k = 1, 2, \dots, n$, and hence obtain a CDR for \mathcal{F} . Approximate evaluation is sufficient for our purpose, since only the value $\lceil x(k) \rceil = \lceil kt(k) \rceil$ is necessary for the construction of CDR. Since $\tau(t, z)$ can be computed in the unit time, we can compute $g_k(t)$. Since $g_k(t)$ is an increasing function, the value $\lceil kt(k) \rceil$ for $t(k) = t_\theta = F_k(\theta)$ can be computed by binary searching over $t \in \{1/k, 2/k, \dots, k - 1/k\}$ to find the value t_0 such that $g(t_0 - 1/k) < \theta \leq g(t_0)$.

We then derandomize the process replacing θ by $V(k)$ for each k .

3.3.2 Upper bound of the Hausdorff distance

We give the analysis for the Hausdorff distance between a curved ray and its digitized ray. We consider the derandomized version here, and the analysis for the randomized version is given later.

For a differentiable curve $C \in \mathcal{F}$, consider the intersection point $p_C(z) = (x_C(z), z - x_C(z))$ with the line $x + y = z$ for $0 < z \leq n$. Let $s_C(z)$ be the slope of C at $p_C(z)$. Our analysis depends on the property of the function $s_C(z)$.

► **Definition 6.** *Given a function $y = f(x)$ defined on an interval I , if I can be decomposed into a minimum number of consecutive subintervals such that $f(x)$ is monotone (either nonincreasing or nondecreasing) on each subinterval, the number of subintervals is called the wave number of f . It is infinity if there is no such decomposition into a finite number of subintervals.*

The wave number of $s_C(z)$ is intuitively the length of the alternating sequence of consecutive convex segments and concave segments of C .

► **Definition 7.** *The wave number of \mathcal{F} is the supremum of the wave numbers of $s_C(z)$ over all $C \in \mathcal{F}$ on the interval $(0, n]$ of z .*

► **Theorem 8.** *If \mathcal{F} is a diffused family of rays with the wave number w , the Hausdorff distance between the ray C going through p in \mathcal{F} and the path $P = S(p)$ from p towards the origin in $T_{\mathcal{F}}^{\text{det}}$ is bounded by $O(\sqrt{wn \log n})$ for any node $p \in G$.*

For the families of parabolas and homogeneous polynomials, we can verify that the wave number is 1, and thus we have Theorems 3 and 4 as corollaries.

In order to prove Theorem 8, we prepare two lemmas. The first one (Lemma 9) is well-known (see e.g. [12]). The area of a planar region X is denoted by $A(X)$.

► **Lemma 9.** *Consider the set of points $S = \{(k, V(k)) : k = 0, 1, 2, \dots, n\}$ in the region $X = [0, n] \times [0, 1]$. Then, for any axis parallel rectangle R in X , the difference (called discrepancy) between the number of points in $S \cap R$ and the area of $A(R)$ is $O(\log n)$.*

The following Lemma 10 gives a discrepancy bound of S with respect to a region below a curve of a function.

► **Lemma 10.** Consider the set of points $S = \{(k, V(k)) : k = 0, 1, 2, \dots, n\}$. Let $f(x)$ be a continuous function from $[0, n]$ to $[0, 1]$ with a wave number w , and let $Q_I(f) = \{(x, y) : 0 \leq y \leq f(x), x \in I\}$ for any given interval $I \subset [0, 1]$. Then, the discrepancy $||S \cap Q_I(f)| - A(Q_I(f))|$ is bounded by $c\sqrt{wn \log n}$ for a suitable constant c .

Proof. Since we can decompose the interval I into w subintervals such that f is monotone on each of them, it suffices to consider the case $w = 1$. Indeed, if subintervals have lengths n_1, n_2, \dots, n_w and has discrepancies $c\sqrt{n_i \log n_i}$ for $i = 1, 2, \dots, w$, the sum $\sum_{i=1}^w c\sqrt{n_i \log n_i}$ is bounded by $c\sqrt{wn \log n}$, where the minimum it attained if $n_i = n/w$ for every i . If $w = 1$, f is either nonincreasing or nondecreasing, and without loss of generality, we assume f is nondecreasing.

We divide $Q_I(f)$ into its intersections with consecutive vertical strips of width $\sqrt{n \log n}$ (possibly the last one is skinnier). Let A_i for $i = 1, 2, \dots, M = \lceil \sqrt{n \log n} \rceil$ be the strips.

Suppose s_i and t_i are x -values of the leftmost and rightmost boundary of A_i , respectively. Now, within the strip A_i , $Q_I(f)$ is contained in a rectangle R_i whose height is $f(t_i)$, and contains another rectangle R'_i whose height is $f(s_i)$. Since $0 \leq f(x) \leq 1$ and f is nondecreasing we can easily see that the difference of areas of $\cup_{i=1}^M R_i$ and $\cup_{i=1}^M R'_i$ is at most the area of a rectangle of height 1 and width $\sqrt{n \log n}$. For a union of M rectangles, we can apply Lemma 9, and the number of points of S in $\cup_{i=1}^M R_i$ is at most $A(Q_I(f)) + \sqrt{n \log n} + O(M \log n)$, and that in $\cup_{i=1}^M R'_i$ is at least $A(Q_I(f)) - \sqrt{n \log n} - O(M \log n)$. Since $M < \sqrt{n \log n} + 1$, we have the lemma. ◀

We remark that for the discrepancy in Lemma 10, an $\Omega(\sqrt{n})$ lower bound is known even if f is a linear function (see [12]).

Now let us give a proof for Theorem 8.

The basic idea is that if P goes too far from C on a level, then it cannot come back to the same destination point p without violating the discrepancy condition given in Lemma 10.

Without loss of generality, we can assume that p is a boundary element located on $L(n)$. For each diagonal $x + y = k$, the intersection of C (resp. P) with it is denoted by $q_C(k) = (x_C(k), y_C(k))$ and $q_P(k) = (x_P(k), y_P(k))$, respectively. Then, the Hausdorff distance is bounded by $\sqrt{2} \max_{1 \leq k \leq n} |x_C(k) - x_P(k)|$, and it suffices to show that there is a constant c' such that $|x_C(k) - x_P(k)| \leq c' \sqrt{wn \log n}$. We take $c' > c$, where c is the constant given in Lemma 10.

Assume on the contrary there exists an index s such that $|x_C(s) - x_P(s)| > c' w \sqrt{n \log n}$. Without loss of generality, we can assume that $x_C(s) > x_P(s)$, since the other case can be handled analogously.

There exists an index m such that $x_C(i) - x_P(i) > 0$ for $s \leq i < m$ and $x_C(m) - x_P(m) \leq 0$ because $x_P(n) = x_C(n)$ (both P and C need to go through p). In other words, the path P lies on the left of C in $L(k)$ for $s \leq k < m$ and first comes back to the (almost) same position on $L(m)$. Let I be the interval $(s, m]$. Thus, we have

$$x_P(m) - x_P(s) > x_C(m) - x_C(s) + c' \sqrt{wn \log n}. \quad (*)$$

In the derandomized construction, $V(k)$ is used (instead of θ) to determine $t(k)$. In our construction method, P has a horizontal incoming edge at $L(k)$ if and only if $g_k(t_P(k)) \geq V(k)$, where $t_P(k) = \frac{x_P(k)}{k}$. By the monotonicity of g_k , $g_k(t_C(k)) \geq g_k(t_P(k))$ if $k \in I$, and this implies $g_k(t_C(k)) \geq V(k)$.

The integer-valued function $x_C(k)$ is extended to a continuous function $x_C(z)$ that gives the x -value of the intersection point of $x + y = z$ and the ray C for a real value $z \in (0, n]$. Moreover, the function $g_k(t) = \frac{1}{1+\tau(t,k)}$ can be extended to $g(t, z) = \frac{1}{1+\tau(t,z)}$.

We define $\varphi_C(z) = g(t_C(z), z) = \frac{1}{1 + \tau(t_C(z), z)}$. Recall that $\tau(t_C(z_0), z_0) = dy/dx|_{z=z_0}$ is the slope of C at $z = z_0$, and hence

$$\varphi_C(z_0) = \frac{1}{1 + \frac{dy}{dx}|_{z=z_0}} = \frac{dx}{dx + dy}|_{z=z_0} = \frac{dx}{dz}|_{z=z_0}.$$

Thus, $\varphi_C(z)$ is the ratio of the increase of $x_C(z)$ to the increase of z in the infinitesimal neighbor of z_0 . The wave number of $\varphi_C(z)$ is the same as that of $\tau(t_C(z), z)$, since $\varphi_C(z)$ is increasing in an interval I if and only if $\tau(t_C(z), z)$ is decreasing. We can observe that $\tau(t_C(z), z) = s_C(z)$ by definition, and hence the wave number of $\varphi_C(z)$ is the same as that of $s_C(z)$, and bounded by w . Also, the range of $\varphi_C(z)$ is in $(0, 1]$.

If the incoming edge of P is horizontal, $\varphi_C(k) = g_k(t_C(k)) \geq V(k)$ as shown above, and this condition is equivalent to $(k, V(k)) \in Q_I(\varphi_C)$, since $Q_I(\varphi_C) = \{(z, x) : 0 \leq x \leq \varphi_C(z), s < z \leq m\}$. Let S be the set of points $(k, V(k))$ for $k = s, s + 1, \dots, m - 1$. The difference of the x -values of P at $z = s$ and $z = m$ is the number of horizontal edges in the interval, which is hence bounded by $|Q_I(\varphi_C) \cap S|$.

On the other hand, $A(Q_I(\varphi_C))$ equals the difference of x -value of C at s and m , since

$$A(Q_I(\varphi_C)) = \int_{s < z < m} \varphi_C(z) dz = \int_{s < z < m} \frac{dx_C(z)}{dz} dz = x_C(m) - x_C(s).$$

Since the wave number of φ_C is bounded by w and its range is in $(0, 1]$, Lemma 10 says that $|Q_I(\varphi_C) \cap S| - A(Q_I(\varphi_C)) < c\sqrt{wn \log n}$. Thus, we have

$$x_P(m) - x_P(s) \leq |Q_I(\varphi_C) \cap S| \leq A(Q_I(\varphi_C)) + c\sqrt{wn \log n} = x_C(m) - x_C(s) + c\sqrt{wn \log n}.$$

Therefore, $x_P(m) - x_P(s) \leq x_C(m) - x_C(s) + c\sqrt{wn \log n}$. Compared with (*), we have $c > c'$, and obtain a contradiction.

3.3.3 Analysis for the randomized version

We would like to mention the quality of the randomized construction of a CDR.

► **Definition 11.** Consider a continuous function f defined on an interval $I = (k, m]$ with the range $[0, 1]$, where $0 < k < m < n$ are positive integers. let \bar{f} be the linear interpolation using the values of f on integer abscissae, which is the piecewise linear curve connecting $(k, f(k)), (k + 1, f(k + 1)), \dots, (m, f(m))$ by linear segments of width 1. The discretization error of f on I is $|A(Q_I(f)) - A(Q_I(\bar{f}))|$.

The following is easy to see.

► **Lemma 12.** If the wave number of f is bounded by w , the discretization error of f is at most w .

Now, given a function f from $[0, n]$ to $[0, 1]$, consider a $\{0, 1\}$ -valued random variable $X_f(i)$ for each $i = 1, 2, \dots, n$ such that it becomes 1 if and only if a uniformly random number (chosen independently for each i) in $[0, 1]$ becomes less than or equals to $f(i)$. Let $X_f = \sum_{i=1}^n X_f(i)$. Then the expected value $E(X) = \sum_{i=1}^n E(X_i)$ equals $\sum_{i=1}^n f(i) = A(Q_I(\bar{f})) + \frac{f(m) - f(k)}{2}$. Note that $A(Q_I(\bar{f})) \leq n$, thus $E(X) \leq n + 1/2$.

We can apply Chernoff's inequality, and obtain a constant $c(r)$ such that $|X - Q(\bar{f})| \geq c(r)\sqrt{n \log n}$ with a probability $1 - n^{-r-3}$ for any given constant r . Now we are ready to analyze the randomized construction of CDR.

► **Theorem 13.** *If \mathcal{F} is a diffused family of rays with the wave number w , the largest Hausdorff distance between a ray and the corresponding digital ray in $T_{\mathcal{F}}^{rand}$ is $O(\sqrt{n} \log n + w)$ with probability $1 - n^{-r}$ for any fixed $r > 0$.*

Proof. Analogously to the deterministic version, For any path p in the CDR corresponding a curve $C = C(p) \in F$, the Hausdorff distance from p to C is bounded by the maximum difference of $A(Q_I(\varphi(C)))$ and $X_{Q_I(\varphi(C))}$ over all I . Since there are $O(n)$ paths from the root to leaves, and there are $O(n^2)$ intervals $[k, m]$, there are $O(n^3)$ choices. Thus, with probability $1 - n^{-r}$, $|A(Q_I(\varphi(\bar{C}(p)))) - X(Q_I(\varphi(C(p))))| \leq c(r)\sqrt{n} \log n$ for all p and I . Thus, we have the theorem. ◀

The above upper bound is worse than the deterministic version by a $\sqrt{\log n}$ factor if w is a small constant, while it is theoretically better if $w > \log n$.

3.4 Family of curves linear in a parameter

Let us consider a nondecreasing differentiable function $y = f(x)$ for $x \in [0, n]$ such that $f(0) = 0$ and $f(x) > 0$ for $x > 0$. We define the family $\mathcal{F} = \{C_a : a \geq 0\}$ of curves, where C_a is defined by $y = af(x)$. It is clear that this gives a ray family.

If C_a goes through (x_0, y_0) , then $a = \frac{y_0}{f(x_0)}$. The slope of the curve C_a at (x_0, y_0) is $af'(x_0)$, which is (eliminating a) $\frac{f'(x_0)y_0}{f(x_0)}$. We consider the slope $\tau(x, k) = \frac{(k-x)f'(x)}{f(x)}$ along the diagonal $x + y = k$ for each k .

If \mathcal{F} is diffused, the framework in the previous subsection works. Although the explicit form of F_k might not be obtained, we can apply binary search to compute $F_k(z)$ for a given z utilizing the monotonicity. Thus, we can compute $x(k) = kF_k(V(k))$ within the pixel precision in $O(\log n)$ time.

Diffusedness and the wave number depend on f . The following lemma is easy to observe.

► **Lemma 14.** *If f is a strictly increasing and concave function, the family \mathcal{F} is diffused, and its wave number is 1.*

Note that the family of rays $\{y = f(a^{-1}x) : a > 0\}$ for a convex function f can be also handled, since this family is $\{x = af^{-1}(y)\}$ and $f^{-1}(y)$ is a concave function, e.g., the families of parabolas and homogeneous polynomials could be regarded in this form. Let us see some typical examples.

► **Example 15.** Let \mathcal{F}_{sig} be the family of curves $y = a\sigma(x)$, $0 \leq a$, where $\sigma(x) = \frac{1}{1+e^{-x}} - \frac{1}{2}$ is the shifted sigmoid function. The curve $y = \sigma(x)$ is strictly increasing and concave; hence, the family is diffused with the wave number 1, and we have the $O(\sqrt{n} \log n)$ bound.

Here, $\tau(x, k) = \frac{(k-x)e^{-x}}{(1+e^{-x})^2\sigma(x)}$. The function $g_k = F_k^{-1}$ can be analytically given, but it is a complicated function such that it is difficult to find an explicit formula for F_k . Thus, we apply the binary searching method to find a value of $F_k(V(k))$ in our experiment.

► **Example 16.** Consider the family of curves $y = a \log(x + 1)$, then similarly we have a CDR with the $O(\sqrt{n} \log n)$ distance bound.

► **Example 17.** The sine curve $y = \sin(x)$ is not monotone. Therefore, we define $\tilde{\sin}(x)$ by $\tilde{\sin}(x) = 0$ for $x < 0$, $\tilde{\sin}(x) = \sin x$ for $0 \leq x \leq \pi/2$ and $\tilde{\sin}(x) = 1$ for $x > \pi/2$. The curve $y = \tilde{\sin}(x)$ is monotonically nondecreasing and differentiable, and we will apply our CDR construction for the family of curves $y = a \tilde{\sin}(x)$ for $a \geq 0$.

Here, the family is weakly-diffused but not diffused, since there are many parallel horizontal lines intersecting each level. However, it is clear that in the region $x > \pi/2$ where the rays becomes horizontal, we can set all edges horizontal. Thus, we can still apply our method to have the $O(\sqrt{n \log n})$ bound.

The obtained CDRs are illustrated in Figure 6 in the section to give experimental results.

4 Union of CDRs with consistency

A CDR is characterized by the sequence $\mathbf{m} : m(1), m(2), \dots, m(n)$ where $1 \leq m(i) = \lceil x(k) \rceil \leq k$, and we denote the CDR by $T(\mathbf{m})$. We denote $\mathbf{m} \succeq \mathbf{m}'$ if $m(i) \geq m'(i)$ for all $1 \leq i \leq n$. \succeq is a partial ordering. We write $\mathbf{m} \succ \mathbf{m}'$ if $\mathbf{m} \succeq \mathbf{m}'$ and $\mathbf{m} \neq \mathbf{m}'$.

Consider $\mathcal{P}(T(\mathbf{m})) \cup \mathcal{P}(T(\mathbf{m}'))$, where $\mathcal{P}(T)$ means the set of paths from the root towards leaf vertices in T .

Let $x_P(k)$ be the x -value of the pixel of a path P (from the root to a leaf) in a CDR on the level $L(k)$.

► **Definition 18.** We say a path P_1 is steeper than another path P_2 in a different CDR if there is an index $0 \leq k \leq n$ such that $x_{P_1}(i) \leq x_{P_2}(i)$ for $i \leq k$ and $x_{P_1}(i) > x_{P_2}(i)$ for $i > k$. We say the level $L(k)$ the break level of P_1 and P_2 .

The above definition implies that P_1 lies below or on P_2 up to the break level, and it lies strictly above P_2 after it. We allow $k = n$, which means P_1 never goes above P_2 . We say the pair of paths have a *singular separation* on a level $L(i)$ if $x_{P_1}(i) = x_{P_2}(i)$ and $x_{P_1}(i+1) > x_{P_2}(i+1)$. Thus, the paths cross each other at most once, although they may touch and singularly separate several times before the break level. We say P_1 and P_2 *semi-consistently intersect* if one is steeper than the other. Moreover, if there is no singular separation, we say they *consistently intersect* each other.

► **Theorem 19.** If $\mathbf{m} \succ \mathbf{m}'$, any path $P_1 \in \mathcal{P}(T(\mathbf{m}))$ is steeper than any path $P_2 \in \mathcal{P}(T(\mathbf{m}'))$. Moreover, if a singular separation happens on a level $L(i)$, $m(i+1) = m'(i+1) = x_{P_1}(i) + 1 = x_{P_2}(i) + 1$.

Proof. Since $\mathbf{m} \succ \mathbf{m}'$, if a vertical edge comes in $p \in L(k)$ in $T_{\mathbf{m}'}$, a vertical edge comes in p in $T_{\mathbf{m}}$, too. This further implies that if such p is located on the right of another pixel q on $L(k)$, every incoming edge to q must be also vertical in both trees.

Therefore, if P_2 lies strictly on the right of P_1 on a level $L(k)$, whenever P_2 selects a vertical incoming edge in $L(k+1)$, P_1 also must select a vertical edge. Thus, inductively the horizontal distance never decreases after the break level, and hence P_1 never meets P_2 again.

Next, we consider what happens at a singular separation. Then, P_1 and P_2 goes through a same point $p = (x, k-x)$ in a level $L(k)$, and P_1 selects a horizontal and P_2 selects a vertical edge towards $L(k+1)$. Then, the vertices of P_1 and P_2 are at positions $q = (x+1, k-x)$ and $q' = (x, k+1-x)$, respectively. Since the incoming edge of $T_{\mathbf{m}}$ to q is horizontal and that of $T_{\mathbf{m}'}$ to q' is vertical, $m(i+1) \leq x+1$ and $m'(i+1) > x$. Since $m(i+1) \geq m'(i+1)$, this happens only if $m(i+1) = m'(i+1) = x+1$. ◀

We note that the condition $m(i+1) = m'(i+1) = x_{P_1} + 1 = x_{P_2} + 1$ for a singular level $L(i)$ means that both $T_{\mathbf{m}}$ and $T_{\mathbf{m}'}$ have the branching node $p = (m(i+1) - 1, i - m(i+1) + 1)$ on the level $L(i)$ simultaneously. Both P_1 and P_2 goes through p , and P_1 selects the horizontal while P_2 selects the vertical branch. We say a node p a *singular point* if it is a shared branching node of $T_{\mathbf{m}}$ a $T_{\mathbf{m}'}$, and hence a singular separation only occurs at a singular point.

A region $R \subseteq \Delta$ is called a *slanted-quadrant* if it is defined as $\{(x, y) \in \Delta \mid x \geq a, y \geq b, x + y \geq c\}$ for nonnegative numbers a, b , and c . We say a set of digital rays semi-consistently (resp. consistently) approximates a family of curves intersecting at most once to each other in a slanted-quadrant R if each pair of digital rays semi-consistently (resp. consistently) intersect each other if they are restricted to $G \cap R$.

Suppose that families \mathcal{F} and \mathcal{F}' has CDRs $T(\mathbf{m})$ and $T(\mathbf{m}')$ for $\mathbf{m} \succ \mathbf{m}'$, respectively. Assume that $\mathcal{F} \cup \mathcal{F}'$ forms a pseudoline arrangement in a slanted quadrant R . Theorem 19 assures that $\mathcal{P}(T(\mathbf{m})) \cup \mathcal{P}(T(\mathbf{m}'))$ consistently approximates $\mathcal{F} \cup \mathcal{F}'$ in R if there is no singular point in R .

4.1 Union of translated copies of a CDR

For the sequence \mathbf{m} and a nonnegative integer s , we define a new sequence \mathbf{m}^s by $m^s(k) = \min(m(k) + s, k)$. Similarly, for a negative integer s , we define \mathbf{m}^s by $m^s(k) = \max(m(k) + s, 1)$. The following lemma is obvious.

► **Lemma 20.** *If $s \geq 1$, $\mathbf{m}^s \succ \mathbf{m}$. If $s \leq -1$, $\mathbf{m} \succ \mathbf{m}^s$.*

► **Theorem 21.** *Suppose that \mathcal{F} and \mathcal{F}^s are ray families digitized by $T(\mathbf{m})$ and $T(\mathbf{m}^s)$, respectively. Assume that $\mathcal{F} \cup \mathcal{F}^s$ forms a pseudoline arrangement in $\Delta_1 : \{(x, y) \in \Delta \mid x \geq 1, y \geq 1\}$. Then $\mathcal{P}(T(\mathbf{m})) \cup \mathcal{P}(T(\mathbf{m}^s))$ consistently approximate $\mathcal{F} \cup \mathcal{F}^s$ in Δ_1 .*

Proof. Semi-consistency is clear from Theorem 19 and Lemma 20. Consider the location of a singular point $p = (m(i+1) - 1, i - m(i+1) + 1)$ for a pair of paths. However, Theorem 19 says that $m(i+1) = m^s(i+1)$. This only happens either $m(i+1) = m^s(i+1) = 1$ or $m(i+1) = m^s(i+1) = i+1$, and hence $p = (0, i)$ or $p = (i, 0)$. Thus the singular points only locate on the coordinate axes. Thus, we have the theorem. ◀

For a CDR $T = T(\mathbf{m})$, we define $\mathcal{U}^K(T) = \cup_{-K \leq i \leq K} \mathcal{P}(T(\mathbf{m}^s))$. It follows from Theorem 21 that $\mathcal{U}^K(T)$ consistently approximates a pseudoline arrangement represented as a union of associated ray families.

► **Example 22 (Consistent digital line arrangement).** Let us consider the family \mathcal{F}_1 of linear rays. Define \mathcal{F}_1^s for $s \geq 0$ (resp. $s \leq -1$) to be the set of rays starting with horizontal (resp. vertical) rays, and continue to linear rays with positive slopes emanating from $(s, -s)$. Let T be the CDR for \mathcal{F}_1 constructed in Section 2.1. Then $\mathcal{U}^K(T)$ consistently digitize $\cup_{-K \leq s \leq K} \mathcal{F}_1^s$ in Δ_1 with the $O(\log n)$ distance bound.

Indeed, for $s > 0$, the structure of $T(\mathbf{m}^s)$ in $\Delta \cap \{(x, y) : x \geq s\}$ is same as the tree obtained by connecting the forest of $T \cap \{(x, y) : y \geq s\}$ by a horizontal path. The case $s < 0$ is similar. Thus, the discrepancy bound remains as same as the one for T .

Note that although we only consider the lines with positive slopes, we can easily mix it with those with negative slopes (obtained by a mirror construction) without losing consistency. The above example shows that we can consistently digitize the line segments in the first quadrant. However, it is weaker than [4] since we only deal with segments on the lines going through $(s, -s)$ for integers s , and we need a finer precision to represent short segments.

► **Example 23 (Consistent digital pseudoline arrangement of shifted parabola rays).** Let us consider the family \mathcal{F}_2 consisting of curves defined by $y = ax^2$ ($a > 0$). Define \mathcal{F}_2^s to be the set of the right halves of parabolas with the apex $(s, -s)$. Then $\mathcal{U}^K(T_{para})$ consistently digitize $\cup_{-K \leq s \leq K} \mathcal{F}_2^s$ in Δ_1 with the $O(\sqrt{n \log n})$ distance bound.

4.2 Union of homogeneous polynomial families

In this section, we assume that the CDRs are constructed deterministically. Let us consider $\mathcal{F}_{i,j} = \mathcal{F}_i \cup \mathcal{F}_j$ for $1 \leq i \leq j$, where \mathcal{F}_i is the family of homogeneous polynomial curves of degree i . Let $\mathbf{m}^{(i)}$ be the sequence (do not confuse this with \mathbf{m}^s given above) corresponding to $T_{\mathcal{F}_i}$ deterministically constructed. Naturally, each pair of curves $f \in \mathcal{F}_i$ and $g \in \mathcal{F}_j$ intersect once in the first quadrant other than the origin, and thus behaves as a pseudoline arrangement in the region $x > 0, y > 0$. We consider the union $\mathcal{T}_{i,j} = \mathcal{P}(T_{\mathcal{F}_i}) \cup \mathcal{P}(T_{\mathcal{F}_j})$ to approximate curves in $\mathcal{F}_{i,j}$.

► **Lemma 24.** $\mathbf{m}^{(j)} \succeq \mathbf{m}^{(i)}$ for $1 \leq i \leq j$.

Proof. Recall that $x(k) = \frac{jkV(k)}{(j-1)V(k)+1}$ in the construction of $T_{\mathcal{F}_j}$. Thus, $m^{(j)}(k) = \lceil \frac{jkV(k)}{(j-1)V(k)+1} \rceil$. Since $\frac{ikV(k)}{(i-1)V(k)+1} \leq \frac{jkV(k)}{(j-1)V(k)+1}$ if $i < j$, we have the lemma. ◀

► **Theorem 25.** $\mathcal{T}_{1,2}$ consistently approximates $\mathcal{F}_{1,2}$ in the region $\{(x, y) \in \Delta \mid x \geq 3, y \geq 3\}$. For $i \geq 2$, $\mathcal{T}_{i,i+1}$ consistently approximates $\mathcal{F}_{i,i+1}$ in the region $R(i) = \{(x, y) \in \Delta \mid x + y \geq 4(i+1)(i+2), x \geq 4i, y \geq 4(i+1)\}$.

Proof. $\mathcal{T}_{i,i+1}$ semi-consistently approximates $\mathcal{F}_{i,i+1}$ in Δ , although the existence of multiple singular points prevents the consistency.

Thus, we study location of singular points to find a subregion R to attain the consistency.

Consider a singular point p in a level $L(k)$. Recall that $p = (m^{(i)}(k+1), k - m^{(i+1)}(k+1))$ and $m^{(i)}(k+1) = m^{(i+1)}(k+1)$ at a singular point p in a level $L(k)$. Since $m^{(i)}(k+1) = m^{(i+1)}(k+1)$, we have

$$\frac{(i+1)(k+1)V(k+1)}{iV(k+1)+1} - \frac{i(k+1)V(k+1)}{(i-1)V(k+1)+1} < 1.$$

For the case $i = 1$, suppose that a singular point appears on $L(k)$, and let $v = V(k+1)$, and $K = k+1$. Then we have $\frac{2Kv}{v+1} - Kv < 1$, which means $Kv(1-v) < v+1$. We assume that $K \geq 6$ and $\frac{3}{K} \leq v \leq 1 - \frac{3}{K}$. Then, $Kv(1-v) \geq 3(1 - \frac{3}{K}) = 3 - \frac{9}{K}$ and hence $3 - \frac{9}{K} < 1 + 1 - \frac{3}{K}$ and hence $K < 6$, and we have contradiction.

Thus, $k+1 < 6$ or $(k+1)v < 3$ or $(k+1)v > k+1-3 = k-2$. The position of the singular point p is $\lceil (k+1)v \rceil, k - \lceil (k+1)v \rceil$, and hence it is located either in the region $x+y \leq 4, x \leq 2$ or $y \leq 2$. Thus, we have the theorem for $\mathcal{T}_{1,2}$.

For $i \geq 2$, $p = (m^{(i)}(k+1), k - m^{(i)}(k+1))$ and $m^{(i)}(k+1) = m^{(i+1)}(k+1)$ at the singular point p . For simplifying the formulas, we set $K = k+1, v = V(k+1)$, and $c = 2(i+1)(i+2)$. We assume $K > 2c$ since otherwise the singular point is outside R .

Now, we have

$$\frac{(i+1)Kv}{iv+1} - \frac{iKv}{(i-1)v+1} < 1.$$

This is transformed to

$$Kv(1-v) < (iv+1)((i+1)v+1). \quad (*)$$

We will first show that it gives a contradiction if $\frac{4}{K} \leq v \leq 1 - \frac{c}{K}$.

For the case where $1/2 \leq v \leq 1 - \frac{c}{K}$, $v(1-v)$ take its minimum at $v = 1 - \frac{c}{K}$, and the right hand side is at most $(i+1)(i+2)$. Thus we have

$$K \frac{c}{K} (1 - \frac{c}{K}) = c(1 - \frac{c}{K}) < (i+1)(i+2)$$

32:14 Digital Curved Rays

Since $K > 2c$, $c(1 - 1/2) < (i + 1)(i + 2)$ and hence $c < 2(i + 1)(i + 2)$ and it contradicts the definition of c .

If $\frac{1}{i+1} \leq v < \frac{1}{2}$, substituting $K > 4(i + 1)(i + 2)$, (*) implies that

$$4(i + 1)(i + 2)v(1 - v) < (iv + 1)((i + 1)v + 1).$$

Cleaning up the formula, we have

$$-(5i^2 + 13i + 8)v^2 + (4i^2 + 10i + 7)v - 1 < 0.$$

Since $v < 1/2$, we replace v^2 by $v/2$, we have

$$-(5i^2 + 13i + 8)\frac{v}{2} + (4i^2 + 10i + 7)v - 1 < 0.$$

And hence $(3i^2 + 7i + 6)v < 2$. This does not happen if $v \geq \frac{1}{i+1}$.

If $\frac{4}{K} \leq v \leq \frac{1}{i+1}$, the left hand of (*) takes minimum at $v = \frac{4}{K}$, and the right hand takes maximum at $v = \frac{1}{i+1}$, and we have

$$4\left(1 - \frac{4}{K}\right) < 2\frac{2i + 1}{i + 1}.$$

Since $K > 4(i + 1)(i + 2)$,

$$1 - \frac{1}{(i + 1)(i + 2)} < 1 - \frac{1}{2(i + 1)}.$$

This does not happen since $i + 2 > 2$.

Thus, we have either $v < \frac{4}{K}$ or $v > 1 - \frac{c}{K}$.

In the former case that $v < \frac{4}{K}$. The x -value of the singular point is $\lceil \frac{iKv}{(i-1)v+1} \rceil \cdot \frac{iKv}{(i-1)v+1}$ is monotonically increasing in v (if $v > 0$). Thus, we have it is less than $\frac{4i}{1+4(i-1)/K}$, which takes maximum at $K = \infty$. Thus, the x -value of the singular point is less than $4i$.

In the latter case, consider the y -value $k - \lceil \frac{(i+1)Kv}{iv+1} \rceil$ of the singular point, which is at most $K - \frac{(i+1)Kv}{iv+1} = \frac{K-Kv}{iv+1}$. It takes the maximum $\frac{c}{i(1-\frac{c}{K})+1}$ at $v = 1 - \frac{c}{K}$, and since $K > 2c$ it is less than $\frac{2c}{i+2} = 4(i + 1)$. ◀

The following is a straightforward corollary.

► **Corollary 26.** $\mathcal{T}_{i,j}$ consistently approximates $\mathcal{F}_{i,j}$ for any $j > i$ in the region $R(i)$. Accordingly, $\mathcal{T}_{\leq d} = \cup_{1 \leq i \leq d} \mathcal{P}(T_i)$ consistently approximates $\mathcal{H}_d = \cup_{1 \leq i \leq d} \mathcal{F}_i$ in the region $R(d - 1)$.

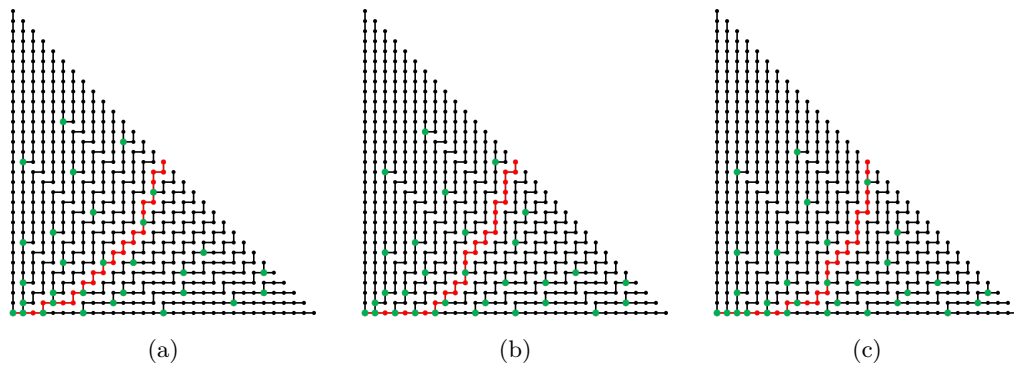
5 Experimental results

We have implemented our method and constructed CDR for the constant-multiplied curves. Figure 5 and Figure 6 illustrate CDRs for polynomial curves, sine, sigmoid, and logarithmic rays.

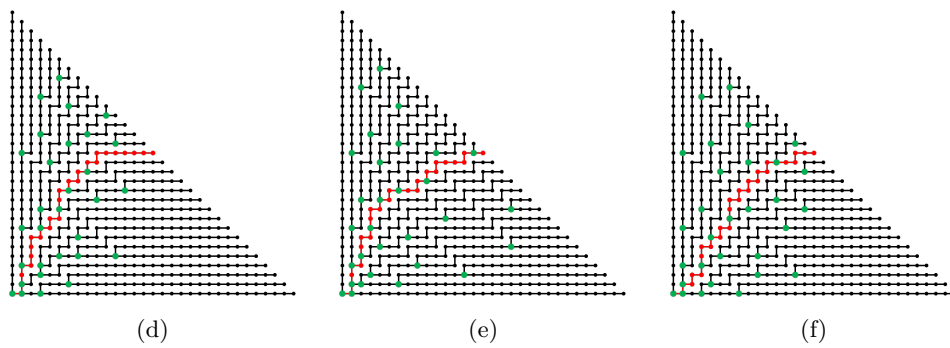
Figure 7 shows the selected paths approximating the curves towards equally-spaced sampled points on the boundary of square regions.

For each grid width $n = 2^m$ up to $n = 2^{14}$, the worst-case Hausdorff distance between parabolas and digital rays in T_{para} is given in Figure 8, where it is about 12 for $n = 2^{14}$. The dependency of worst-case Hausdorff distance on n have the similar behavior for each of other types of curve.

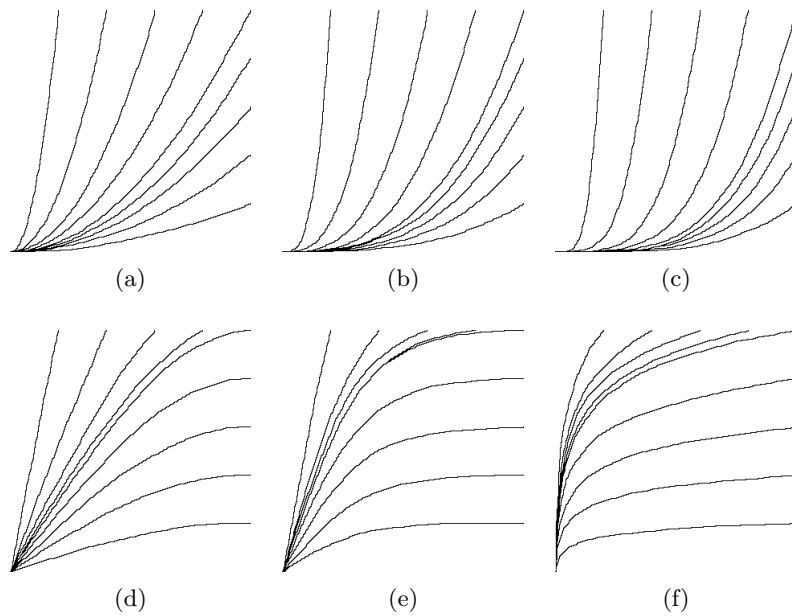
The error for $n = 2^{14}$ is about 11.2, 13.4, 15.0 for sine, sigmoid and logarithmic curves, respectively. Note that the values are real numbers since we consider Hausdorff distance based on Euclidean distance.



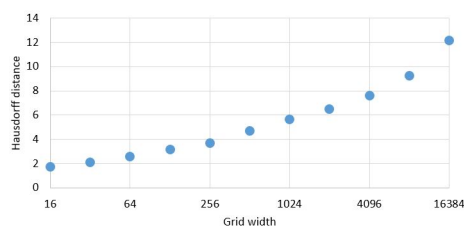
■ **Figure 5** CDRs for (a) $y = ax^2$, (b) $y = ax^3$, and (c) $y = ax^4$. Green nodes are branching nodes. Red paths are the digital curves towards $p = (15, 15)$.



■ **Figure 6** CDRs for (d) $y = a\tilde{\sin}x$ ($0 \leq x \leq \pi/2$), (e) $y = a\sigma(x)$ ($x \leq 6$), and (f) $y = a \log(x + 1)$. Green nodes are branching nodes. Red paths are the digital curves towards $p = (15, 15)$.



■ **Figure 7** Sampled Curves of CDRs for (a) $y = ax^2$, (b) $y = ax^3$, (c) $y = ax^4$, (d) $y = a\tilde{\sin}x$ ($0 \leq x \leq \pi/2$), (e) $y = a\sigma(x)$ ($x \leq 6$), and (f) $y = a \log(x + 1)$ in the 300×300 grid.



■ **Figure 8** The largest distance from a parabola and the corresponding digital ray in T_{para} .

6 Concluding remarks

The experimental result suggests that our $O(\sqrt{n \log n})$ bound seems to be loose. Although currently the lower bound mentioned for Lemma 10 prevents us to improve it beyond $O(\sqrt{n})$, recent progress on low-discrepancy sequences [16] might be applied.

For the line segments, a construction of consistent digital segments (CDS) is known [4] with $O(\log n)$ distance error bound. Although we have a generalization of CDS to handle some families of curves, there are a lot of questions to invest further: For example, we do not know how to handle the set of all axis parallel parabolas.

References

- 1 M. Chiu, M. Korman, M. Sutherland, and T. Tokuyama. Distance bounds for 3-D consistent digital rays and 2-D partially-consistent digital rays. In preparation.
- 2 I. Chowdhury and M Gibson. A Characterization of Consistent Digital Line Segments in \mathbb{Z}^2 . In *Proc. 23rd ESA*, pages 337–348, 2015.
- 3 I. Chowdhury and M Gibson. Constructing Consistent Digital Line Segments. In *Proc. 12th LATIN*, pages 263–274, 2016.
- 4 T. Christ, D. Pálvölgyi, and M. Stojaković. Consistent Digital Line Segment. *Discrete & Computational Geometry*, 47-4:691–710, 2012.
- 5 J. Chun, M. Korman, M. Nöllenburg, and T. Tokuyama. Consistent Digital Rays. *Discrete & Computational Geometry*, 42-3:359–378, 2009.
- 6 F.P.Preparata and M.I.Shamos. *Computational Geometry – an Introduction*. Springer Verlag, 1985.
- 7 D. Greene and F. Yao. Finite Resolutional Computational Geometry. In *Proc. 27th IEEE FOCS*, pages 143–152, 1986.
- 8 D. Halperin and E. Packer. Iterated Snap Rounding. *Comput. Geom. Theor. Appl.*, 23:209–225, 2002.
- 9 J. Hershberger. Stable Snap Rounding. *Comput. Geom. Theor. Appl.*, 46:403–416, 2013.
- 10 J. Hobby. Practical Segment Intersection with Finite Precision Output. *Comput. Geom. Theor. Appl.*, 13:199–214, 1999.
- 11 R. Klette and A. Rosenfeld. Digital straightness – a review. *Discrete Applied Math.*, 139:197–230, 2004.
- 12 J. Matoušek. *Geometric Discrepancy*. Springer Verlag, 1991.
- 13 K. Mehlhorn and S. Näher. *LEDA: a platform for combinatorial and geometric computation*. Cambridge University Press, 1999.
- 14 M.G.Lubby. Grid geometries which preserve properties of Euclidean geometry: A study of graphics line drawing algorithms. In *NATO Conference on Graphics/CAD*, pages 397–432, 1987.
- 15 V. Milenkovic. Double Precision Geometry: A General Technique for Calculating Lines and Segment Intersections using Rounding Arithmetic. In *Proc. 30th IEEE FOCS*, pages 500–505, 1989.
- 16 H. Niederreiter. Recent constructions of low-discrepancy sequences. *Mathematics and Computers in Simulation*, 135:18–27, 2017.

Efficient Approximation Schemes for Uniform-Cost Clustering Problems in Planar Graphs

Vincent Cohen-Addad

Sorbonne Université, CNRS, Laboratoire d'informatique de Paris 6, LIP6, F-75252 Paris, France
vincent.cohen.addad@ens-lyon.org

Marcin Pilipczuk

Institute of Informatics, University of Warsaw, Poland
marcin.pilipczuk@mimuw.edu.pl

Michał Pilipczuk

Institute of Informatics, University of Warsaw, Poland
michal.pilipczuk@mimuw.edu.pl

Abstract

We consider the k -MEDIAN problem on planar graphs: given an edge-weighted planar graph G , a set of clients $C \subseteq V(G)$, a set of facilities $F \subseteq V(G)$, and an integer parameter k , the task is to find a set of at most k facilities whose opening minimizes the total connection cost of clients, where each client contributes to the cost with the distance to the closest open facility. We give two new approximation schemes for this problem:

- *FPT Approximation Scheme*: for any $\varepsilon > 0$, in time $2^{\mathcal{O}(k\varepsilon^{-3} \log(k\varepsilon^{-1}))} \cdot n^{\mathcal{O}(1)}$ we can compute a solution that has connection cost at most $(1 + \varepsilon)$ times the optimum, with high probability.
- *Efficient Bicriteria Approximation Scheme*: for any $\varepsilon > 0$, in time $2^{\mathcal{O}(\varepsilon^{-5} \log(\varepsilon^{-1}))} \cdot n^{\mathcal{O}(1)}$ we can compute a set of at most $(1 + \varepsilon)k$ facilities whose opening yields connection cost at most $(1 + \varepsilon)$ times the optimum connection cost for opening at most k facilities, with high probability.

As a direct corollary of the second result we obtain an EPTAS for UNIFORM FACILITY LOCATION on planar graphs, with same running time.

Our main technical tool is a new construction of a “coreset for facilities” for k -MEDIAN in planar graphs: we show that in polynomial time one can compute a subset of facilities $F_0 \subseteq F$ of size $k \cdot (\log n / \varepsilon)^{\mathcal{O}(\varepsilon^{-3})}$ with a guarantee that there is a $(1 + \varepsilon)$ -approximate solution contained in F_0 .

2012 ACM Subject Classification Theory of computation → Facility location and clustering; Theory of computation → Fixed parameter tractability

Keywords and phrases k -Median, Facility Location, Planar Graphs, Approximation Scheme

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.33

Related Version Full version of the paper accessible at <http://arxiv.org/abs/1905.00656>.

Funding This work is a part of projects CUTACOMBS (Ma. Pilipczuk) and TOTAL (Mi. Pilipczuk) that have received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreements No. 714704 and No. 677651, respectively). Ce projet a bénéficié d’une aide de l’État gérée par l’Agence Nationale de la Recherche au titre du Programme Appel à projets générique JCJC 2018 portant la référence suivante: ANR-18-CE40-0004-01.



1 Introduction

We study approximation schemes for classic clustering objectives, formalized as follows. Given an edge-weighted graph G together with a set C of vertices called *clients*, a set F of vertices called *candidate facilities*, and an *opening cost* $\text{open} \in \mathbb{R}_{\geq 0}$, the UNIFORM FACILITY LOCATION problem asks for a subset of facilities (also called centers) $D \subseteq F$ that minimizes



© Vincent Cohen-Addad, Marcin Pilipczuk, and Michał Pilipczuk;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, 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

the cost defined as $|D| \cdot \text{open} + \sum_{c \in C} \min_{f \in D} \text{dist}(c, f)$. In the NON-UNIFORM FACILITY LOCATION variant, the opening costs may vary between facilities.

We also consider the related k -MEDIAN problem, where the tuple (G, F, C) comes with a hard budget k for the number of open facilities (as opposed to the opening cost `open`). That is, the problem asks for a set $D \subseteq F$ of size at most k that minimizes the connection cost $\sum_{c \in C} \text{dist}(c, D)$. Note that UNIFORM FACILITY LOCATION can be reduced to k -MEDIAN by guessing the number of open facilities in an optimal solution.

FACILITY LOCATION and k -MEDIAN model in an abstract way various clustering objectives appearing in applications. Therefore, designing approximation algorithms for them and their variants is a vibrant topic in the field of approximation algorithms. For NON-UNIFORM FACILITY LOCATION, a long line of work [1, 15, 22, 16] culminated with the 1.488-approximation algorithm by Li [18]. On the other hand, Guha and Khuller [13] showed that the problem cannot be approximated in polynomial time within factor better than 1.463 unless $\text{NP} \subseteq \text{DTIME}[n^{\mathcal{O}(\log \log n)}]$, which gives almost tight bounds on the best approximation factor achievable in polynomial time. For k -MEDIAN, the best known approximation ratio achievable in polynomial time is 2.67 due to Byrka et al. [3], while the lower bound of 1.463 due to Guha and Khuller [13] holds here as well.

Given the approximation hardness status presented above, it is natural to consider restricted metrics. In this work we consider *planar metrics*: we assume that the underlying edge-weighted graph G is planar.

It was a long-standing open problem whether FACILITY LOCATION admits a polynomial-time approximation scheme (PTAS) in planar metrics. For the uniform case, this question has been resolved in affirmative by Cohen-Addad et al. [8] in an elegant way: they showed that local search of radius $\mathcal{O}(1/\varepsilon^2)$ actually yields a $(1 + \varepsilon)$ -approximation, giving a PTAS with running time $n^{\mathcal{O}(1/\varepsilon^2)}$. This approach also gives a PTAS for k -MEDIAN with a similar running time, and works even in metrics induced by graphs from any fixed proper minor-closed class.

Very recently, Cohen-Addad et al. [9] also gave a PTAS for NON-UNIFORM FACILITY LOCATION in planar metrics using a different approach. Roughly, the idea is to first apply Baker layering scheme to reduce the problem to the case when in all clusters (sets of clients connected to the same facility) in the solution, all clients are within distance between 1 and r from the center, for some constant r depending only on ε . This case is then resolved by another application of Baker layering scheme, followed by a dynamic programming on a hierarchical decomposition of the graph using shortest paths as balanced separators.

Both the schemes of [8] and of [9] are PTASes: they run in time $n^{g(\varepsilon)}$ for some function g . It is therefore natural to ask for an *efficient PTAS* (EPTAS): an approximation scheme with running time $f(\varepsilon) \cdot n^{\mathcal{O}(1)}$ for some function f . Such an EPTAS was given by Cohen-Addad [5] for k -MEANS in low-dimensional Euclidean spaces; this is a variant of k -MEDIAN where every client contributes to the connection cost with the *square* of its distance from the closest open facility. Here, the idea is to apply local search as in [8], but to use the properties of the metric to explore the local neighborhood faster. Unfortunately, this technique mainly relies on the Euclidean structure (or on the bounded doubling dimension of the input) and seems hard to lift to the general planar case. Also the techniques of [9] are far from yielding an EPTAS: essentially, one needs to use a logarithmic number of portals at every step of the final dynamic programming in order to tame the accumulation of error through $\log n$ levels of the decomposition. Recently, a near-linear time EPTAS was given by Cohen-Addad et al. [6] for k -median, k -means and non-uniform facility location where the input points lie in a metric of bounded doubling dimension. This result is obtained using a randomized dissection of the metric, a technique that is not available for planar inputs.

The goal of this work is to circumvent these difficulties and give an EPTAS for UNIFORM FACILITY LOCATION in planar metrics.

Our results. Our main technical contribution is the following theorem. In essence, it states that when solving k -MEDIAN on a planar graph one can restrict the facility set to a subset of size $k \cdot (\varepsilon^{-1} \log n)^{\mathcal{O}(\varepsilon^{-3})}$, at the cost of losing a multiplicative factor of $(1 + \varepsilon)$ on the optimum connection cost. This can be seen as the planar version of the classic result by Matoušek [20] who showed that for Euclidean metrics of dimension d , it is possible to reduce the number of candidate centers to $\text{poly}(k)\varepsilon^{-\mathcal{O}(d)}$ at the cost of losing a multiplicative factor of $(1 + \varepsilon)$ on the optimum connection cost (through the use of coresets as well). For general metrics, obtaining such a result seems challenging, since this would imply a $(1 + \varepsilon)$ -approximation algorithm with running time $f(k, \varepsilon)n^{\mathcal{O}(1)}$, which would contradict Gap-ETH [7].

From now on, by *with constant probability* we mean with probability at least $1/2$; this can be boosted by independent repetition.

► **Theorem 1.** *Given a k -Median instance (G, F, C, k) , where G is a planar graph, and an accuracy parameter $\varepsilon > 0$, one can in randomized polynomial time compute a set $F_0 \subseteq F$ of size $k \cdot (\varepsilon^{-1} \log n)^{\mathcal{O}(\varepsilon^{-3})}$ satisfying the following condition with constant probability: there exists a set $D_0 \subseteq F_0$ of size at most k such that for every set $D \subseteq F$ of size at most k it holds that $\sum_{c \in C} \min_{u \in D_0} \text{dist}(u, c) \leq (1 + \varepsilon) \sum_{c \in C} \min_{u \in D} \text{dist}(u, c)$.*

A direct corollary of Theorem 1 is a fixed-parameter approximation scheme for the k -MEDIAN problem in planar graphs. This continues the line of work on fixed-parameter approximation schemes for k -median and k -means in Euclidean spaces [12, 17], where the goal is to design an algorithm running in time $f(k, \varepsilon) \cdot n^{\mathcal{O}(1)}$ for a computable function f .

► **Corollary 2.** *Given a k -Median instance (G, F, C, k) , where G is a planar graph, and an accuracy parameter $\varepsilon > 0$, one can in randomized time $2^{\mathcal{O}(k\varepsilon^{-3} \log(k\varepsilon^{-1}))} \cdot n^{\mathcal{O}(1)}$ compute a solution $D \subseteq F$ that has connection cost at most $(1 + \varepsilon)$ times the minimum possible connection cost with constant probability.*

Proof. Apply the algorithm of Theorem 1 and let $F_0 \subseteq F$ be the obtained subset of facilities. Then run a brute-force search through all subsets of F_0 of size at most k and output one with the smallest connection cost. Thus, the running time is

$$\left(k \cdot (\varepsilon^{-1} \log n)^{\mathcal{O}(\varepsilon^{-3})}\right)^k \cdot n^{\mathcal{O}(1)} \leq 2^{\mathcal{O}(k\varepsilon^{-3} \log(k\varepsilon^{-1}))} \cdot (\log n)^{\mathcal{O}(k\varepsilon^{-3})} \cdot n^{\mathcal{O}(1)}$$

which is at most $2^{\mathcal{O}(k\varepsilon^{-3} \log(k\varepsilon^{-1}))} \cdot n^{\mathcal{O}(1)}$ since $(\log n)^d \leq 2^{\mathcal{O}(d \log d)} \cdot n^{\mathcal{O}(1)}$: if $n \leq 2^{d^2}$ then $(\log n)^d \leq d^{2d} \leq 2^{\mathcal{O}(d \log d)}$, and if $n > 2^{d^2}$ then $(\log n)^d \leq 2^{\sqrt{\log n} \cdot \log \log n} \leq n^{\mathcal{O}(1)}$. ◀

Using Theorem 1 we can also give an efficient bicriteria PTAS for k -MEDIAN in planar graphs. This time, the proof is more involved and uses the local search techniques of [5].

► **Theorem 3.** *Given a k -Median instance (G, F, C, k) , where G is a planar graph, and an accuracy parameter $\varepsilon > 0$, one can in randomized time $2^{\mathcal{O}(\varepsilon^{-5} \log(\varepsilon^{-1}))} \cdot n^{\mathcal{O}(1)}$ compute a set $D \subseteq F$ of size at most $(1 + \varepsilon)k$ such that its connection cost is at most $(1 + \varepsilon)$ times the minimum possible connection cost for solutions of size k with constant probability.*

A direct corollary of Theorem 3 is an efficient PTAS for UNIFORM FACILITY LOCATION in planar graphs.

► **Theorem 4.** *Given a Uniform Facility Location instance (G, F, C, open) , where G is a planar graph, and an accuracy parameter $\varepsilon > 0$, one can in randomized time $2^{\mathcal{O}(\varepsilon^{-5} \log(\varepsilon^{-1}))} \cdot n^{\mathcal{O}(1)}$ compute a solution $D \subseteq F$ that has total cost at most $(1 + \varepsilon)$ times the optimum cost with constant probability.*

Proof. Iterate over all possible choices of k being the number of facilities opened by the optimum solution, and for every k invoke the algorithm of Theorem 3 for the k -MEDIAN instance (G, F, C, k) . From the obtained solutions output one with the smallest cost. ◀

Note that the approach presented above fails for the non-uniform case, where each facility has its own, distinct opening cost.

In this extended abstract we focus on proving the main result, Theorem 1. The proof of Theorem 3, on which Theorem 4 also relies, is fully deferred to the full version of the paper [10].

Our techniques. The first step in the proof of Theorem 1 is to reduce the number of relevant clients using the coreset construction of Feldman and Langberg [11]. By applying this technique, we may assume that there are at most $k \cdot \mathcal{O}(\varepsilon^{-2} \log n)$ clients in the instance, however they are weighted: every client c is assigned a nonnegative weight $\omega(c)$, and it contributes to the connection cost of any solution with $\omega(c)$ times the distance to the closest open facility in the solution.

We now examine the Voronoi diagram induced in the input graph G by the *clients*: vertices of G are classified into *cells* according to the closest client. This Voronoi diagram has one cell per every client, thus it can be regarded as a planar graph with $|C|$ faces, where each face accommodates one cell. To formally define the Voronoi diagram, and in particular the boundaries between neighboring cells, we use the framework introduced by Marx and Pilipczuk [19] and its extension used in [21].

Consider now all the *spokes* in the diagram, where a spoke is the shortest path connecting the center of a cell (i.e. a client) with a branching node of the diagram incident to the cell (which is a face of G). Removing all the spokes and all the branching nodes from the plane divides it into *diamonds*, where each diamond is delimited by four spokes, called further the *perimeter* of the diamond. See Figure 1 for an example. Since the diagram is a planar graph with $|C|$ faces, there are $\mathcal{O}(|C|) = k \cdot \mathcal{O}(\varepsilon^{-2} \log n)$ diamonds altogether. Moreover, since no diamond contains a client in its interior, whenever P is a path connecting a client with a facility belonging to some diamond Δ , P has to cross the perimeter of Δ .

Now comes the key and most technical part of the proof. We very carefully put $\mathcal{O}(\varepsilon^{-2} \log n)$ portals on the perimeter of each diamond. The idea of placement is similar to that of the resolution metric used in the QPTAS for FACILITY LOCATION. Namely, on a spoke Q starting at client c we put portals at distance $1, (1 + \varepsilon), (1 + \varepsilon)^2, \dots$ from c , so that the further we are on the spoke from c , the sparser the portals are. As a diamond is delimited by four spokes, we may thus use only $\mathcal{O}(\varepsilon^{-2} \log n)$ portals per diamond, while the cost of snapping a path crossing Q to the portal closest to the crossing point can be bounded by ε times the distance from the crossing point to c .

For a facility f in a diamond Δ , we define the *profile* of f as follows. For every spoke Q in the perimeter of Δ , we look at the closest portal π from f on Q . We record approximate (up to $(1 + \varepsilon)$ multiplicative error) distances from f to π and $\mathcal{O}(\varepsilon^{-3})$ neighboring portals, as well as the distance to the client endpoint of the spoke Q . The crux lies in the following fact: for every two facilities f, f' in Δ with the same profile, replacing f with f' increases the connection cost of any client c connected to f only by a multiplicative factor of $(1 + \varepsilon)$. Hence, for every profile in every diamond it suffices to keep just one facility with that profile. Since there are $k \cdot \mathcal{O}(\varepsilon^{-2} \log n)$ diamonds and $\mathcal{O}(\varepsilon^{-1} \log n)^{\mathcal{O}(\varepsilon^{-3})}$ possible profiles in each of them, we keep at most $k \cdot (\varepsilon^{-1} \log n)^{\mathcal{O}(\varepsilon^{-3})}$ facilities in total. This proves Theorem 1.

For the proof of Theorem 3, we first apply Theorem 1 to reduce the number of facilities to $k \cdot (\varepsilon^{-1} \log n)^{\mathcal{O}(\varepsilon^{-3})}$. Then we again inspect the Voronoi diagram, but now induced by the *facilities*. Having contracted every cell to a single vertex, we compute an r -division of the

obtained planar graph to cover it with regions of size $r = (\varepsilon^{-1} \log n)^{\mathcal{O}(\varepsilon^{-3})}$ so that only $\mathcal{O}(\varepsilon)k$ facilities are on boundaries of the regions. We open all the facilities in all the boundaries – thus exceeding the quota for open facilities by $\mathcal{O}(\varepsilon)k$ – run the PTAS of Cohen-Addad et al. [8] in each region independently, and at the end assemble regional solutions using a knapsack dynamic programming. Since within each region there are only polylogarithmically many facilities, each application of the PTAS actually works in time $f(\varepsilon) \cdot n^{\mathcal{O}(1)}$.

2 Preliminaries on Voronoi diagrams and coresets

In this section we recall some tools about Voronoi diagrams in planar graphs and coresets that will be used in the proof of Theorem 1. We will consider undirected graphs with positive edge lengths embedded in a sphere, with the standard shortest-paths metric $\text{dist}(u, v)$ for $u, v \in V(G)$. Contrary to the previous section, the metric is defined on the vertex set of G only, i.e., we do not consider G as a metric space with points in the interiors of edges. For $X, Y \subseteq V(G)$, we denote $\text{dist}(X, Y) = \min_{x \in X, y \in Y} \text{dist}(x, y)$ and similarly we define $\text{dist}(u, X)$ for $u \in V(G)$ and $X \subseteq V(G)$.

Recall that for a set $D \subseteq V(G)$ of *open facilities* and a set $C \subseteq V(G)$, we define the *connection cost* as

$$\text{conn}(D, C) = \sum_{v \in C} \text{dist}(v, D).$$

If the input is additionally equipped with opening costs $\text{open}: F \rightarrow \mathbb{R}_{\geq 0}$, then the *opening cost* of D is defined as $\sum_{w \in D} \text{open}(w)$.

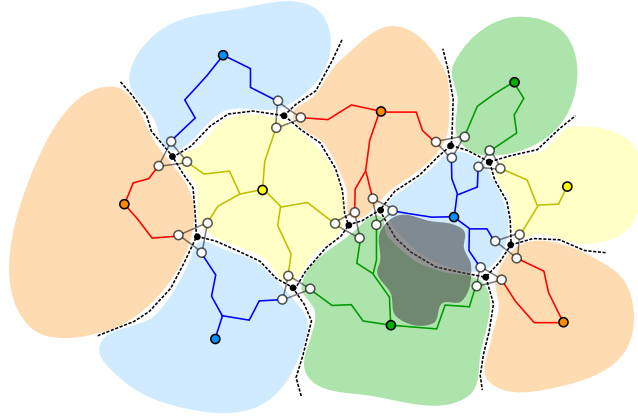
2.1 Voronoi diagrams in planar graphs

We now recall the construction of Voronoi diagrams and related notions in planar graphs used by Marx and Pilipczuk [19]. The setting is as follows. Suppose G is an n -vertex simple graph embedded in a sphere Σ whose edges are assigned nonnegative real lengths. We consider the shortest path metric in G : for two vertices u, v , their *distance* $\text{dist}(u, v)$ is equal to the smallest possible total length of a path from u to v . We will assume that G is triangulated (i.e. every face of G is a cycle of length 3), for this may always be achieved by triangulating the graph using edges of infinite weight.

Further, we assume that shortest paths are unique in G and that finite distances between distinct vertices in G are pairwise different: for all vertices u, v, u', v' with $u \neq v$, $u' \neq v'$ and $\{u, v\} \neq \{u', v'\}$, we have $\text{dist}(u, v) \neq \text{dist}(u', v')$ or $\text{dist}(u, v) = \text{dist}(u', v') = +\infty$. This can be achieved by adding small perturbations to the edge lengths. Since we never specify degrees of polynomials in the running time of our algorithms, we may ignore the additional complexity cost incurred by the need of handling the perturbations in arithmetic operations.

Voronoi diagrams and their properties. Suppose that S is a subset of vertices¹ of G . First, define the *Voronoi partition*: for a vertex $p \in S$, the *Voronoi cell* $\text{Cell}_S(p)$ is the set of all those vertices $u \in V(G)$ whose distance from p is smaller than the distance from any other vertex $q \in S$; note that ties do not occur due to the distinctness of distances in G . Note that $\{\text{Cell}_S(p)\}_{p \in S}$ is a partition of the vertex set of G . For each $p \in S$, let $T(p)$ be the union of

¹ In [19] a more general setting is considered where objects inducing the diagram are connected subgraphs of G instead of single vertices. We will not need this generality here.



■ **Figure 1** A part of the Voronoi diagram with various features distinguished. Branching nodes of the diagram are grayed triangular faces, edges of the diagram are dashed. Solid paths of respective colors are spokes. (The interior of) one diamond is grayed in order to highlight it.

shortest paths from vertices of $\text{Cell}_S(p)$ to p ; recall here that shortest paths in G are unique. Note that, due to the distinctness of distances in G , $T(p)$ is a spanning tree of the subgraph of G induced by the cell $\text{Cell}_S(p)$.

The diagram Vor_S induced by G is a multigraph constructed as follows. First, take the dual G^* of G and remove all edges dual to the edges of all the trees $T(p)$, for $p \in S$. Then, exhaustively remove vertices of degree 1. Finally, for every maximal 2-path (i.e. path with internal vertices of degree 2), say with endpoints u and v , we replace this path by the edge uv ; note that this creates a loop at u in case $u = v$. The resulting multigraph Vor_S is the Voronoi diagram induced by S . Note that the vertices of Vor_S are faces of G ; for clarity we shall call them *branching nodes*. Furthermore, Vor_S inherits an embedding in Σ from the dual G^* , where an edge uv that replaced a maximal 2-path P is embedded precisely as P , i.e., as the concatenation of (the embeddings of) the edges comprising P . From now on we will assume this embedding of Vor_S .

We recall several properties of Vor_S , observed in [19]

► **Lemma 5** (Lemmas 4.4 and 4.5 of [19]). *The diagram Vor_S is a connected and 3-regular multigraph embedded in Σ , which has exactly $|S|$ faces, $2|S| - 4$ branching nodes, and $3|S| - 6$ edges. The faces of Vor_S are in one-to-one correspondence with vertices of S : each $p \in S$ corresponds to a face of Vor_S that contains all vertices of $\text{Cell}_S(p)$ and no other vertex of G .*

Spokes and diamonds. We now introduce further structural elements that can be distinguished in the Voronoi diagram, see Figure 1 for reference. The definitions and basic observations presented below are taken from Pilipczuk et al. [21], and were inspired by the Euclidean analogues due to Har-Peled [14].

An *incidence* is a triple $\tau = (p, u, f)$ where $p \in S$, f is a branching node of the diagram Vor_S , and u is a vertex of G that lies on f (recall that f is a triangular face of G) and belongs to $\text{Cell}_S(p)$. The *spoke* of the incidence τ , denoted $\text{Spoke}(\tau)$, is the shortest path in G between p and u . Note that all the vertices of $\text{Spoke}(\tau)$ belong to $\text{Cell}_S(p)$.

Let $e = f_1 f_2$ be an edge of the diagram Vor_S , where f_1, f_2 are branching nodes (possibly $f_1 = f_2$ if e is a loop in Vor_S). Further, let p_1 and p_2 be the vertices from S that correspond to faces of Vor_S incident to e (possibly $p_1 = p_2$ if e is a bridge in Vor_S). Suppose for a moment that $f_1 \neq f_2$. Then, out of the three edges of f_1 (these are edges in G) there is

exactly one that crosses the edge e of Vor_S ; say it is the edge $u_{1,1}u_{1,2}$ where $u_{1,1} \in \text{Cell}_S(p_1)$ and $u_{1,2} \in \text{Cell}_S(p_2)$. Symmetrically, there is one edge of f_2 that crosses e , say it is $u_{2,1}u_{2,2}$ where $u_{2,1} \in \text{Cell}_S(p_1)$ and $u_{2,2} \in \text{Cell}_S(p_2)$. In case $f_1 = f_2$, the edge e crosses two different edges of $f_1 = f_2$ and we define $u_{1,1}, u_{1,2}, u_{2,1}, u_{2,2}$ analogously for these two crossings; note that then, provided p_1 corresponds to the face enclosed by the loop e , we have $u_{1,1} = u_{1,2}$. For all $i, j \in \{1, 2\}$, consider the incidence $\tau_{i,j} = (p_i, u_{i,j}, f_j)$.

Consider removing the following subsets from the sphere Σ : interiors of faces f_1, f_2 and spokes $\text{Spoke}(\tau_{i,j})$ for all $i, j \in \{1, 2\}$. After this removal the sphere breaks into two regions, out of which exactly one, say R , intersects (the embedding of) e . Let the *diamond* of e , denoted $\text{Diam}(e)$, be the subgraph of G consisting of all features (vertices and edges) embedded in $R \cup \bigcup_{i,j \in \{1,2\}} \text{Spoke}(\tau_{i,j})$. The region R as above is the *interior* of the diamond $\text{Diam}(e)$. Note that in particular, the spokes $\text{Spoke}(\tau_{i,j})$ for $i, j \in \{1, 2\}$ and the edges $u_{1,1}u_{1,2}$ and $u_{2,1}u_{2,2}$ belong to $\text{Diam}(e)$. The *perimeter* of the diamond of e is the closed walk obtained by concatenating spokes $\text{Spoke}(\tau_{i,j})$ for $i, j \in \{1, 2\}$ and edges $u_{1,1}u_{1,2}, u_{2,1}u_{2,2}$ in the natural order around $\text{Diam}(e)$. The following observation is immediate:

► **Proposition 6.** *Consider removing all the spokes (considered as curves on Σ) and all the branching nodes (considered as interiors of faces on Σ) of the diagram Vor_S from the sphere Σ . Then Σ breaks into $3|S| - 6$ regions that are in one-to-one correspondence with edges of Vor_S : a region corresponding to the edge e is the interior of the diamond $\text{Diam}(e)$. Consequently, the intersection of diamonds of two different edges of Vor_S is contained in the intersection of their perimeters.*

Finally, we note that the perimeter of a diamond separates it from the rest of the graph. Since vertices of S are never contained in the interior of a diamond, this yields the following.

► **Lemma 7.** *Let $p \in S$ and u be a vertex of G belonging to the diamond $\text{Diam}(e)$ for some edge e of Vor_S . Then every path in G connecting u and p intersects the perimeter of $\text{Diam}(e)$.*

2.2 Coresets

In most our algorithms, the starting point is the notion of a *coreset* and a corresponding result of Feldman and Langberg [11]. To this end, we need to slightly generalize the notion of a client set in a k -MEDIAN instance. A *client weight function* is a function $\omega: C \rightarrow \mathbb{R}_{\geq 0}$. Given a set $D \subseteq F$ of open facilities, the (weighted) connection cost is defined as

$$\text{conn}(D, \omega) = \sum_{v \in C} \text{dist}(v, D) \cdot \omega(v).$$

That is, every client v is assigned a weight $\omega(v)$ with which it contributes to the objective function. The *support* of a weight function ω is defined as $\text{supp}(\omega) = \{v \in C \mid \omega(v) > 0\}$. From now on, whenever we speak about a k -MEDIAN instance without specified client weight function, we assume the standard function assigning each client weight 1.

The essence of coresets is that one can find weight functions with small support that well approximate the original instance. Given a k -MEDIAN instance (G, F, C, k) (without weights) and an accuracy parameter $\varepsilon > 0$, a *coreset* is a weight function ω such that for every set $D \subseteq F$ of size at most k , it holds that

$$|\text{conn}(D, C) - \text{conn}(D, \omega)| \leq \varepsilon \cdot \text{conn}(D, C).$$

We rely on the following result of Feldman and Langberg [11].

► **Theorem 8** (Theorem 15.4 of [11]). *Given a k -Median instance (G, F, C, k) with $n = |V(G)|$ and accuracy parameter $\varepsilon > 0$, one can in randomized polynomial time find a weight function ω with support of size $\mathcal{O}(k\varepsilon^{-2} \log n)$ that is a coreset with constant probability.*

We note that Ke Chen [4] gave a construction of a strong coreset with support of size $\mathcal{O}(k^2\varepsilon^{-2} \log n)$ that is much simpler than the later construction of Feldman and Langberg [11]. By using this construction instead, we would obtain a weaker version of Theorem 1, with a bound on $|F_0|$ that is quadratic in k instead of linear. This would be perfectly sufficient to derive an FPT approximation scheme as in Corollary 2, but for Theorem 3 we will vitally use the stronger statement. A construction of coresets with similar size guarantees, but maintainable in the streaming model, has been proposed by Braverman et al. [2].

3 Facility coreset for k -Median in planar graphs

In this section we give a coreset for centers for the k -MEDIAN problem, that is, we prove Theorem 1. We shall focus on the following lemma, which in combination with Theorem 8 yields Theorem 1.

► **Lemma 9.** *Given a k -Median instance (G, F, C, k) with a weight function ω and an accuracy parameter $\varepsilon > 0$, one can in polynomial time compute a set $F_0 \subseteq F$ of size $|\text{supp}(\omega)| \cdot (\varepsilon^{-1} \log |V(G)|)^{\mathcal{O}(\varepsilon^{-3})}$ satisfying the following condition with constant probability: there exists a set $D_0 \subseteq F_0$ of size at most k such that for every set $D \subseteq F$ of size at most k it holds that $\text{conn}(D_0, \omega) \leq (1 + \varepsilon) \cdot \text{conn}(D, \omega)$.*

Before we proceed, let us verify that Theorem 8 and Lemma 9 together imply Theorem 1. Given an instance (G, F, C, k) of k -MEDIAN, we first apply Theorem 8 to obtain a coreset ω with support of size $\mathcal{O}(k\varepsilon^{-2} \log n)$. Next, we pass this coreset to Lemma 9, thus obtaining a set $F_0 \subseteq F$ of size $k \cdot (\varepsilon^{-1} \log n)^{\mathcal{O}(\varepsilon^{-3})}$. Let D_0 be the subset of F_0 of size at most k that minimizes $\text{conn}(D_0, \omega)$. Then using the approximation guarantees of Theorem 8 and Lemma 9, for any $D \subseteq F$ we have

$$\text{conn}(D_0, C) \leq (1 + \varepsilon)\text{conn}(D_0, \omega) \leq (1 + \varepsilon)^2\text{conn}(D, \omega) \leq (1 + \varepsilon)^3\text{conn}(D, C).$$

It remains to rescale ε . Hence, for the rest of this section we focus on proving Lemma 9.

Let $\mathcal{I} = (G, F, C, k)$ be an input k -MEDIAN instance with a weight function ω , where G is planar. Let $\varepsilon > 0$ be an accuracy parameter and without loss of generality assume that $\varepsilon < 1/4$. Let $n = |V(G)|$ and $m = |E(G)|$. Without loss of generality assume that $n = \Theta(m)$.

We assume that G is embedded in a sphere Σ and apply the necessary modifications explained in the beginning of Section 2.1 to fit into the framework of Voronoi diagrams. Denote $S = \text{supp}(\omega)$. We compute the Voronoi partition Cell_S induced by S and the Voronoi diagram Vor_S induced by S . By Proposition 6, Vor_S has $\mathcal{O}(|S|)$ vertices, faces, and edges.

Distance levels. We first compute an $\mathcal{O}(1)$ -approximate solution $\tilde{D} \subseteq F$ using the algorithm given by Feldman and Langberg [11, Theorem 15.1]; this algorithm outputs an $\mathcal{O}(1)$ -approximate solution with constant probability. Let us scale all the edge lengths in G by the same ratio so that

$$\text{conn}(\tilde{D}, \omega) = |S|/\varepsilon. \tag{1}$$

Next, we assign length $+\infty$ to every edge of length larger than $\text{conn}(\tilde{D}, \omega)$; clearly, they are not used in the computation of the connection cost of an optimum solution. Without loss of

generality we assume that all the distances between vertices in G are finite: otherwise we can split the instance into a number of independent ones, compute a suitable set F_0 for each of them and take the union.

The next step is to assign levels to distances in the graph. For any $c \in [0, +\infty)$, define the *level* of c , denoted $\text{level}(c)$, to be the smallest nonnegative integer ℓ such that $c < (1 + \varepsilon)^\ell$. Note that $\text{level}(c) = 0$ if and only if $c < 1$. Let $L = 1 + \text{level}(m \cdot \text{conn}(\tilde{D}, \omega))$, then we have

$$\text{level}(\text{dist}(u, v)) \in \{0, 1, \dots, L - 1\} \quad \text{for all } u, v \in V(G).$$

Observe that since $m = \Theta(n)$, by (1) we have

$$L \leq \mathcal{O}(\varepsilon^{-1} \log(m|S|/\varepsilon)) \leq \mathcal{O}(\varepsilon^{-2} \log n). \quad (2)$$

Portals and profiles. Let $\tau = (p, u, f)$ be an incidence in Vor_S . Let $d(\tau) = \text{dist}(p, u)$ and let $\ell(\tau) = \text{level}(d(\tau))$; note that $\text{Spoke}(\tau)$ has length exactly $d(\tau)$. For every integer $\iota \in \{1, \dots, \ell(\tau)\}$, we define the *portal* $\mathbf{p}\langle\tau, \iota\rangle$ as a vertex on $\text{Spoke}(\tau)$ at distance exactly $(1 + \varepsilon)^{\iota-1}$ from p ; we subdivide an edge and create a new vertex to accommodate $\mathbf{p}\langle\tau, \iota\rangle$ if necessary. Furthermore, we add also a portal $\mathbf{p}\langle\tau, 0\rangle = p$. Since $\ell(\tau) = \text{level}(d(\tau)) < L$, there are at most L portals on the spoke $\text{Spoke}(\tau)$.

Consider a diamond $\text{Diam}(e)$ induced by some edge e of Vor_S , and a vertex w in $\text{Diam}(e)$. Recall that the perimeter of $\text{Diam}(e)$ consists of spokes $\text{Spoke}(\tau_{i,j})$ for four incidence $\tau_{i,j}$, where $i, j \in \{1, 2\}$. The *profile* of a vertex w belonging to the diamond $\text{Diam}(e)$ consists of the following information, for all $\tau \in \{\tau_{i,j} : i, j \in \{1, 2\}\}$:

1. The minimum index $\lambda \in \{0, 1, \dots, \ell(\tau)\}$ satisfying

$$\text{dist}(\mathbf{p}\langle\tau, \lambda\rangle, p) > \varepsilon \cdot \text{dist}(\mathbf{p}\langle\tau, \lambda\rangle, w),$$

where $p = \mathbf{p}\langle\tau, 0\rangle$ is the vertex of S involved in τ . If no such index exists, we set $\lambda = \ell(\tau)$.

2. Letting

$$I = (\{0\} \cup \{\iota : |\iota - \lambda| \leq 1/\varepsilon^3\}) \cap \{0, 1, \dots, \ell(\tau)\},$$

the profile records the value of $\text{level}(\text{dist}(w, \mathbf{p}\langle\tau, \iota\rangle))$ for all $\iota \in I$.

Whenever speaking about a vertex w and incidence τ , we use $\lambda(\tau, w)$ and $I(\tau, w)$ to denote λ and I as above. We note that in total there are only few possible profiles.

▷ **Claim 10.** The number of possible different profiles of vertices in $\text{Diam}(e)$ is $L^{\mathcal{O}(\varepsilon^{-3})}$.

Proof. Since $0 \leq \ell(\tau) < L$ for every incidence τ , there are at most L^4 choices for the four values $\lambda(\tau_{i,j}, w)$ for $i, j \in \{1, 2\}$. Further, we have $|I(\tau_{i,j}, w)| \leq \mathcal{O}(\varepsilon^{-3})$, so there are at most $L^{\mathcal{O}(\varepsilon^{-3})}$ choices for the values $\text{level}(\text{dist}(w, \mathbf{p}\langle\tau_{i,j}, \iota\rangle))$ for $i, j \in \{1, 2\}$ and $\iota \in I(\tau_{i,j}, w)$. ◁

For future reference, we state the key property of profiles: having the same profile implies having approximately same distances to the profiles with indices in I .

▷ **Claim 11.** Suppose w and w' are two vertices of $\text{Diam}(e)$ that have the same profile. Then for each $\tau \in \{\tau_{i,j} : i, j \in \{1, 2\}\}$ and $\iota \in I(\tau, w)$, we have

$$\text{dist}(w', \mathbf{p}\langle\tau, \iota\rangle) \leq (1 + \varepsilon) \cdot \text{dist}(w, \mathbf{p}\langle\tau, \iota\rangle) + 1.$$

Proof. Let $\ell = \text{level}(\text{dist}(w, \mathbf{p}\langle\tau, \iota\rangle)) = \text{level}(\text{dist}(w', \mathbf{p}\langle\tau, \iota\rangle))$, as recorded in the common profile. If $\ell = 0$, then $\text{dist}(w', \mathbf{p}\langle\tau, \iota\rangle) < 1$ and we are done. Otherwise, $\text{dist}(w, \mathbf{p}\langle\tau, \iota\rangle)$ and $\text{dist}(w', \mathbf{p}\langle\tau, \iota\rangle)$ are both contained in the interval $[(1 + \varepsilon)^{\ell-1}, (1 + \varepsilon)^\ell]$. This interval has length $\varepsilon \cdot (1 + \varepsilon)^{\ell-1} \leq \varepsilon \cdot \text{dist}(w, \mathbf{p}\langle\tau, \iota\rangle)$, hence the claim follows. ◁

33:10 Efficient Approximation Schemes for Planar Clustering

Construction of the set F_0 . We now construct the set F_0 as follows: for every diamond $\text{Diam}(e)$ and every possible profile in $\text{Diam}(e)$, include in F_0 one facility with that profile (if one exists). Since there are $\mathcal{O}(|S|)$ diamonds, by Claim 10 and (10) we have

$$|F_0| \leq \mathcal{O}(|S|) \cdot L^{\mathcal{O}(\varepsilon^{-3})} = |\text{supp}(\omega)| \cdot (\varepsilon^{-1} \log n)^{\mathcal{O}(\varepsilon^{-3})},$$

as claimed. It remains to prove that F_0 has the claimed approximation properties.

For every facility $w \in F$, pick a diamond $\text{Diam}(e)$ containing w and let $f(w)$ to be the facility $f(w) \in F_0 \cap \text{Diam}(e)$ that has the same profile as w . Fix a solution $D^* \subseteq F$ with $|D^*| \leq k$ minimizing $\text{conn}(D^*, \omega)$. Let $D_0 = \{f(w) : w \in D^*\}$. Clearly, $|D_0| \leq |D^*| \leq k$. To finish the proof of Lemma 9 it suffices to show that

$$\text{conn}(D_0, \omega) \leq (1 + \mathcal{O}(\varepsilon))\text{conn}(D^*, \omega). \quad (3)$$

To this end, consider any client $v \in S = \text{supp}(\omega)$ and let $w \in D^*$ be the facility in D^* serving v , that is, $\text{dist}(v, w) = \text{dist}(v, D^*)$. To show (3), it suffices to prove that

$$\text{dist}(v, f(w)) \leq (1 + \mathcal{O}(\varepsilon))\text{dist}(v, w) + \mathcal{O}(1). \quad (4)$$

Indeed, by summing (4) through all $v \in S$ and using (1) we obtain

$$\begin{aligned} \text{conn}(D_0, \omega) &\leq (1 + \mathcal{O}(\varepsilon))\text{conn}(D^*, \omega) + \mathcal{O}(1) \cdot |S| \\ &\leq (1 + \mathcal{O}(\varepsilon))\text{conn}(D^*, \omega) + \mathcal{O}(\varepsilon) \cdot \text{conn}(\tilde{D}, \omega) \leq (1 + \mathcal{O}(\varepsilon))\text{conn}(D^*, \omega), \end{aligned}$$

where the last inequality is due to \tilde{D} being an $\mathcal{O}(1)$ -approximate solution.

Hence, from now on we focus on proving (4). Let $\text{Diam}(e)$ be the diamond containing w and $f(w)$. Consider the shortest path P from w to v in G . By Lemma 7, the path P intersects the perimeter of the diamond $\text{Diam}(e)$. Let u be the vertex on the perimeter of $\text{Diam}(e)$ that lies on P and, among such, is closest to w on P . Since P is a shortest path, the length of the subpaths of P between v and u and between u and w equal $\text{dist}(v, u)$ and $\text{dist}(u, w)$, respectively, and in particular $\text{dist}(v, w) = \text{dist}(v, u) + \text{dist}(u, w)$.

We now observe that to prove (4), it suffices show the following.

$$\text{dist}(u, f(w)) \leq \text{dist}(u, w) + \mathcal{O}(\varepsilon)\text{dist}(v, w) + \mathcal{O}(1). \quad (5)$$

Indeed, assuming (5) we have

$$\begin{aligned} \text{dist}(v, f(w)) &\leq \text{dist}(v, u) + \text{dist}(u, f(w)) \\ &\leq \text{dist}(v, u) + \text{dist}(u, w) + \mathcal{O}(\varepsilon)\text{dist}(v, w) + \mathcal{O}(1) \\ &= \text{dist}(v, w) + \mathcal{O}(\varepsilon)\text{dist}(v, w) + \mathcal{O}(1). \end{aligned}$$

Hence, from now on we focus on proving (5).

Let $\tau_{i,j}$ for $i, j \in \{1, 2\}$, be the four incidences involved in the diamond $\text{Diam}(e)$. Since u lies on the perimeter of $\text{Diam}(e)$, actually u lies on $\text{Spoke}(\tau)$, where $\tau = \tau_{i,j}$ for some $i, j \in \{1, 2\}$. Let $p = \mathbf{p}\langle \tau, 0 \rangle$ be the vertex of S involved in the incidence τ . Since $u \in \text{Cell}_S(p)$ while $v \in S$, we have that $\text{dist}(u, p) \leq \text{dist}(u, v)$. Consequently, we have $\text{dist}(u, w) \leq \text{dist}(v, w)$ and $\text{dist}(u, p) \leq \text{dist}(v, w)$, so to prove (5) it suffices to prove the following:

$$\text{dist}(u, f(w)) \leq \text{dist}(u, w) + \mathcal{O}(\varepsilon)(\text{dist}(u, w) + \text{dist}(u, p)) + \mathcal{O}(1). \quad (6)$$

Let $\mathbf{p}_u = \mathbf{p}\langle \tau, \iota \rangle$ be the portal on the subpath of $\text{Spoke}(\tau)$ between u and p that is closest to u . Intuitively, \mathbf{p}_u is a good approximation of u and distances from \mathbf{p}_u are almost the same as distances from u . As this idea will be repeatedly used in this sequel, we encapsulate it in a single claim.

▷ Claim 12. Suppose for some vertices x and y we have

$$\text{dist}(\mathbf{p}_u, x) \leq A \cdot \text{dist}(\mathbf{p}_u, y) + B,$$

for some A, B . Then

$$\text{dist}(u, x) \leq A \cdot \text{dist}(u, y) + B + (A + 1)\varepsilon \cdot \text{dist}(u, p) + (A + 1).$$

Proof. By the choice of \mathbf{p}_u we have

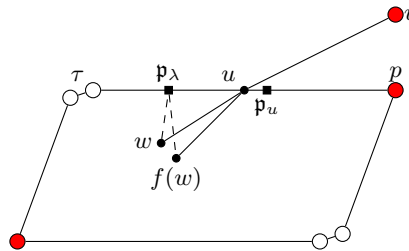
$$\text{dist}(\mathbf{p}_u, p) \leq \text{dist}(u, p) \leq (1 + \varepsilon)\text{dist}(\mathbf{p}_u, p) + 1,$$

so $\text{dist}(u, \mathbf{p}_u) \leq \varepsilon \cdot \text{dist}(u, p) + 1$. Therefore, we have

$$\begin{aligned} \text{dist}(u, x) &\leq \text{dist}(\mathbf{p}_u, x) + \text{dist}(\mathbf{p}_u, u) \leq A \cdot \text{dist}(\mathbf{p}_u, y) + B + \text{dist}(\mathbf{p}_u, u) \\ &\leq A \cdot (\text{dist}(u, y) + \text{dist}(\mathbf{p}_u, u)) + B + \text{dist}(\mathbf{p}_u, u) \\ &= A \cdot \text{dist}(u, y) + B + (A + 1) \cdot \text{dist}(\mathbf{p}_u, u) \\ &\leq A \cdot \text{dist}(u, y) + B + (A + 1) \cdot \varepsilon \cdot \text{dist}(u, p) + (A + 1), \end{aligned}$$

as claimed. ◁

Since w and $f(w)$ have the same profile, we may denote $\lambda = \lambda(\tau, w) = \lambda(\tau, f(w))$ and $I = I(\tau, w) = I(\tau, f(w))$. Further, let $\mathbf{p}_\lambda = \mathbf{p}(\tau, \lambda)$. We now consider a number of cases depending on the relative values of ι and λ , with the goal on proving that (6) holds in each case. See Figure 2 for an illustration.



■ **Figure 2** The diamond $\text{Diam}(e)$ with vertices v, u, w, p , and $f(w)$. Red vertices are clients, black squares are portals. The case distinction in the proof corresponds to relative order of \mathbf{p}_λ and \mathbf{p}_ι .

Middle case: $\iota \in I$. As profiles of w and $f(w)$ are the same and $\iota \in I$, by Claim 11 we have

$$\text{dist}(\mathbf{p}_u, f(w)) \leq (1 + \varepsilon)\text{dist}(\mathbf{p}_u, w) + 1.$$

It suffices now to apply Claim 12 to infer inequality (6).

Close case: $\iota < \lambda - 1/\varepsilon^3$. Since $\iota < \lambda - 1/\varepsilon^3$, by the choice of λ we have

$$\text{dist}(\mathbf{p}_u, p) \leq \varepsilon \cdot \text{dist}(\mathbf{p}_u, w).$$

By applying Claim 12, we infer that

$$\text{dist}(u, p) \leq \varepsilon \cdot \text{dist}(u, w) + (1 + \varepsilon)\varepsilon \cdot \text{dist}(u, p) + \mathcal{O}(1) \leq \varepsilon \cdot \text{dist}(u, w) + \text{dist}(u, p)/2 + \mathcal{O}(1),$$

33:12 Efficient Approximation Schemes for Planar Clustering

which entails

$$\text{dist}(u, p) \leq 2\varepsilon \cdot \text{dist}(u, w) + \mathcal{O}(1).$$

Since $0 \in I$ and the profiles of w and $f(w)$ are equal, by Claim 11 we have

$$\text{dist}(p, f(w)) \leq (1 + \varepsilon) \cdot \text{dist}(p, w) + 1.$$

By combining the last two inequalities, we conclude that

$$\begin{aligned} \text{dist}(u, f(w)) &\leq \text{dist}(u, p) + \text{dist}(p, f(w)) \\ &\leq \text{dist}(u, p) + (1 + \varepsilon)\text{dist}(p, w) + 1 \\ &\leq \text{dist}(u, p) + (1 + \varepsilon)(\text{dist}(u, p) + \text{dist}(u, w)) + 1 \\ &= (2 + \varepsilon)\text{dist}(u, p) + (1 + \varepsilon)\text{dist}(u, w) + 1 \\ &\leq (1 + \mathcal{O}(\varepsilon))\text{dist}(u, w) + \mathcal{O}(1). \end{aligned}$$

Thus, inequality (6) holds in this case.

Far case: $\iota > \lambda + 1/\varepsilon^3$. By the definition of λ and since $\iota > \lambda + 1/\varepsilon^3$, we have in particular $\lambda < \ell(\tau)$ and hence

$$\varepsilon \cdot \text{dist}(\mathbf{p}_\lambda, w) < \text{dist}(\mathbf{p}_\lambda, p).$$

On the other hand, we have

$$\text{dist}(p, \mathbf{p}_\lambda) = (1 + \varepsilon)^{\lambda - \iota} \text{dist}(p, \mathbf{p}_u) \leq (1 + \varepsilon)^{-1/\varepsilon^3} \text{dist}(p, \mathbf{p}_u) \leq \varepsilon^2 \cdot \text{dist}(p, \mathbf{p}_u),$$

where the last step follows from Bernoulli's inequality. By combining the above two inequalities we obtain

$$\text{dist}(\mathbf{p}_\lambda, w) \leq \varepsilon \cdot \text{dist}(p, \mathbf{p}_u),$$

implying

$$\text{dist}(p, w) \leq \text{dist}(p, \mathbf{p}_\lambda) + \text{dist}(\mathbf{p}_\lambda, w) \leq (\varepsilon + \varepsilon^2) \cdot \text{dist}(p, \mathbf{p}_u) \leq 2\varepsilon \cdot \text{dist}(p, \mathbf{p}_u) \leq 2\varepsilon \cdot \text{dist}(p, u).$$

As before, by Claim 11 we have $\text{dist}(p, f(w)) \leq (1 + \varepsilon)\text{dist}(p, w) + 1$ due to $0 \in I$, hence

$$\begin{aligned} \text{dist}(u, f(w)) &\leq \text{dist}(u, p) + \text{dist}(p, f(w)) \\ &\leq (\text{dist}(u, w) + \text{dist}(p, w)) + (1 + \varepsilon)\text{dist}(p, w) + 1 \\ &\leq \text{dist}(u, w) + (2 + \varepsilon) \cdot (2\varepsilon) \cdot \text{dist}(u, p) + 1 \\ &\leq \text{dist}(u, w) + \mathcal{O}(\varepsilon)\text{dist}(u, p) + \mathcal{O}(1). \end{aligned}$$

We conclude that inequality (6) holds in this case.

As the case investigation presented above covers all the possibilities, the proof of Lemma 9 is complete, so we have also proved Theorem 1.

References

- 1 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local Search Heuristics for k -Median and Facility Location Problems. *SIAM J. Comput.*, 33(3):544–562, 2004.
- 2 Vladimir Braverman, Dan Feldman, and Harry Lang. New Frameworks for Offline and Streaming Coreset Constructions. *CoRR*, abs/1612.00889, 2016. [arXiv:1612.00889](#).
- 3 Jarosław Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An Improved Approximation for k -Median and Positive Correlation in Budgeted Optimization. *ACM Trans. Algorithms*, 13(2):23:1–23:31, 2017.
- 4 Ke Chen. On Coresets for k -Median and k -Means Clustering in Metric and Euclidean Spaces and Their Applications. *SIAM J. Comput.*, 39(3):923–947, 2009.
- 5 Vincent Cohen-Addad. A Fast Approximation Scheme for Low-dimensional k -means. In *SODA 2018*, pages 430–440. SIAM, 2018.
- 6 Vincent Cohen-Addad, Andreas Emil Feldmann, and David Saulpic. Near-Linear Time Approximation Schemes for Clustering in Doubling Metrics. *CoRR*, abs/1812.08664, 2018. [arXiv:1812.08664](#).
- 7 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT Approximations for k -Median and k -Means. *ICALP’19*, 2019.
- 8 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 *FOCS 2016*, pages 353–364. IEEE Computer Society, 2016.
- 9 Vincent Cohen-Addad, Marcin Pilipczuk, and Michał Pilipczuk. A Polynomial-Time Approximation Scheme for Facility Location on Planar Graphs. In *FOCS 2019*, 2019.
- 10 Vincent Cohen-Addad, Marcin Pilipczuk, and Michał Pilipczuk. Efficient approximation schemes for uniform-cost clustering problems in planar graphs. *CoRR*, abs/1905.00656, 2019. [arXiv:1905.00656](#).
- 11 Dan Feldman and Michael Langberg. A Unified Framework for Approximating and Clustering Data. *CoRR*, abs/1106.1379, 2011. Conference version presented at STOC 2011. [arXiv:1106.1379](#).
- 12 Wenceslas Fernandez de la Vega, Marek Karpinski, Claire Kenyon, and Yuval Rabani. Approximation schemes for clustering problems. In *STOC 2003*, pages 50–58, 2003.
- 13 Sudipto Guha and Samir Khuller. Greedy Strikes Back: Improved Facility Location Algorithms. *J. Algorithms*, 31(1):228–248, 1999.
- 14 Sarel Har-Peled. Quasi-Polynomial Time Approximation Scheme for Sparse Subsets of Polygons. In *SoCG 2014*. ACM, 2014.
- 15 Dorit S. Hochbaum. Heuristics for the fixed cost median problem. *Math. Program.*, 22(1):148–162, 1982.
- 16 K. Jain and V. Vazirani. Approximation algorithms for metric facility location and k -Median problems using the primal-dual schema and Lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.
- 17 Amit Kumar, Yogish Sabharwal, and Sandeep Sen. Linear-time approximation schemes for clustering problems in any dimensions. *J. ACM*, 57(2), 2010.
- 18 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Inf. Comput.*, 222:45–58, 2013.
- 19 Dániel Marx and Michał Pilipczuk. Optimal Parameterized Algorithms for Planar Facility Location Problems Using Voronoi Diagrams. In *ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 865–877. Springer, 2015. See [arXiv:1504.05476](#) for the full version.
- 20 Jiří Matoušek. On Approximate Geometric k -Clustering. *Discrete & Computational Geometry*, 24(1):61–84, 2000.

33:14 Efficient Approximation Schemes for Planar Clustering

- 21 Michał Pilipczuk, Erik Jan van Leeuwen, and Andreas Wiese. Quasi-Polynomial Time Approximation Schemes for Packing and Covering Problems in Planar Graphs. In *ESA 2018*, volume 112 of *LIPICs*, pages 65:1–65:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
- 22 David B Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems. In *STOC 1997*, pages 265–274. ACM, 1997.

Improved Bounds for the Excluded-Minor Approximation of Treedepth

Wojciech Czerwiński

Institute of Informatics, University of Warsaw, Poland
wczerin@mimuw.edu.pl

Wojciech Nadara

Institute of Informatics, University of Warsaw, Poland
W.Nadara@mimuw.edu.pl

Marcin Pilipczuk

Institute of Informatics, University of Warsaw, Poland
malcin@mimuw.edu.pl

Abstract

Treedepth, a more restrictive graph width parameter than treewidth and pathwidth, plays a major role in the theory of sparse graph classes. We show that there exists a constant C such that for every integers $a, b \geq 2$ and a graph G , if the treedepth of G is at least $Cab \log a$, then the treewidth of G is at least a or G contains a subcubic (i.e., of maximum degree at most 3) tree of treedepth at least b as a subgraph.

As a direct corollary, we obtain that every graph of treedepth $\Omega(k^3 \log k)$ is either of treewidth at least k , contains a subdivision of full binary tree of depth k , or contains a path of length 2^k . This improves the bound of $\Omega(k^5 \log^2 k)$ of Kawarabayashi and Rossman [SODA 2018].

We also show an application for approximation algorithms of treedepth: given a graph G of treedepth k and treewidth t , one can in polynomial time compute a treedepth decomposition of G of width $\mathcal{O}(kt \log^{3/2} t)$. This improves upon a bound of $\mathcal{O}(kt^2 \log t)$ stemming from a tradeoff between known results.

The main technical ingredient in our result is a proof that every tree of treedepth d contains a subcubic subtree of treedepth at least $d \cdot \log_3((1 + \sqrt{5})/2)$.

2012 ACM Subject Classification Mathematics of computing \rightarrow Graph theory

Keywords and phrases treedepth, excluded minor

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.34

Related Version A full version of the paper is available at <https://arxiv.org/abs/1904.13077>.

Funding This research is a part of a project that have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme Grant Agreement 714704.



1 Introduction

For an undirected graph G , the *treedepth* of G is the minimum height of a rooted forest whose ancestor-descendant closure contains G as a subgraph. Together with more widely known related width notions such as treewidth and pathwidth, it plays a major role in structural graph theory, in particular in the study of general sparse graph classes [8, 7, 6].

An important property of treedepth is that it admits a number of equivalent definitions. Following the definition of treedepth above, a *treedepth decomposition* of a graph G consists of a rooted forest F and an injective mapping $f : V(G) \rightarrow V(F)$ such that for every $uv \in E(G)$ the vertices $f(u)$ and $f(v)$ are in ancestor-descendant relation in F . The *width* of a treedepth decomposition (F, f) is the height of F (the number of vertices on the longest leaf-to-root path in F) and the treedepth of G is the minimum possible height of a treedepth decomposition



© Wojciech Czerwiński, Wojciech Nadara, and Marcin Pilipczuk;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 34; pp. 34:1–34:13



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of G . A *centered coloring* of a graph G is an assignment $\alpha : V(G) \rightarrow \mathbb{Z}$ such that for every connected subgraph H of G , α has a *center* in H : a vertex $v \in V(H)$ of unique color, i.e., $\alpha(v) \neq \alpha(w)$ for every $w \in V(H) \setminus \{v\}$. A *vertex ranking* of a graph G is an assignment $\alpha : V(G) \rightarrow \mathbb{Z}$ such that in every connected subgraph H of G there is a unique vertex of *maximum* rank (value $\alpha(v)$). Clearly, each vertex ranking is a centered coloring. It turns out that the minimum number of colors (minimum size of the image of α) needed for a centered coloring and for a vertex ranking are equal and equal to the treedepth of a graph [6].

While there are multiple examples of algorithmic usage of treedepth in the theory of sparse graphs [7, 8], our understanding of the complexity of computing minimum width treedepth decompositions is limited. For a graph G , let $\text{td}(G)$ and $\text{tw}(G)$ denote the treedepth and the treewidth of G , respectively. An algorithm of Reidl, Rossmanith, Villaamil, and Sikdar [9] computes exactly the treedepth of an input graph G in time $2^{\mathcal{O}(\text{td}(G) \cdot t)} n^{\mathcal{O}(1)}$, given a tree decomposition of G of width t . Combined with the classic constant-factor approximation algorithm for treewidth that runs in $2^{\mathcal{O}(\text{tw}(G))} n^{\mathcal{O}(1)}$ time [10], one obtains an exact algorithm for treedepth running in time $2^{\mathcal{O}(\text{td}(G)\text{tw}(G))} n^{\mathcal{O}(1)}$. No faster exact algorithm is known.

For approximation algorithms, the following folklore lemma (presented with full proof in [4]) is very useful.

► **Lemma 1.** *Given a graph G and a tree decomposition (T, β) of G of maximum bag size w , one can in polynomial time compute a treedepth decomposition of G of width at most $w \cdot \text{td}(T)$.*

Using Lemma 1, one can obtain an approximation algorithm for treedepth with a cheap tradeoff trick.¹

► **Lemma 2.** *Given a graph G , one can in polynomial time compute a treedepth decomposition of G of width $\mathcal{O}(\text{td}(G) \cdot \text{tw}(G)^2 \log \text{tw}(G))$.*

Proof. Let $n = |V(G)|$. Using the polynomial-time approximation algorithm for treewidth [3], compute a tree decomposition (T, β) of G of width $t = \mathcal{O}(\text{tw}(G) \sqrt{\log \text{tw}(G)})$ and $\mathcal{O}(n)$ bags. For every integer $1 \leq k \leq (\log n)/t$, use the algorithm of [9] to check in polynomial time if the treedepth of G is at most k . Note that if this is the case, the algorithm finds an optimal treedepth decomposition and we can conclude. Otherwise, we have $\log n \leq \text{td}(G) \cdot t$ and we apply Lemma 1 to G and (T, β) obtaining a treedepth decomposition of G of width

$$\mathcal{O}(t \log n) \leq \mathcal{O}(\text{td}(G) \cdot t^2) \leq \mathcal{O}(\text{td}(G) \cdot \text{tw}(G)^2 \log \text{tw}(G)). \quad \blacktriangleleft$$

Lemma 2 is the only polynomial approximation algorithm for treedepth running in polynomial time we are aware of.

A related topic to exact and approximation algorithms computing minimum-width treedepth decomposition is the study of obstructions to small treedepth. Dvořák, Giannopoulou, and Thilikos [2] proved that every minimal graph of treedepth k has the number of vertices at most double-exponential in k . More recently, Kawarabayashi and Rossman showed an excluded-minor theorem for treedepth.

► **Theorem 3** ([4]). *There exists a universal constant C such that for every integer k every graph of treedepth at least $Ck^5 \log^2 k$ is either of treewidth at least k , contains a subdivision of a full binary tree of depth k as a subgraph, or contains a path of length 2^k .*

¹ This trick has been observed and communicated to us by Michał Pilipczuk. We thank Michał for allowing us to include it in this paper.

While neither the results of [2] nor [4] have a direct application in the approximability of treedepth, these topics are tightly linked with each other and we expect that a finer understanding of treedepth obstructions is necessary to provide more efficient algorithms computing or approximating the treedepth of a graph.

Our results

Our main result is the following statement, improving upon the work of Kawarabayashi and Rossman [4].

► **Theorem 4.** *Let G be a graph of treewidth $\text{tw}(G)$ and treedepth $\text{td}(G)$. Then there exists a subcubic tree H that is a subgraph of G and is of treedepth $\Omega(\text{td}(G)/(\text{tw}(G) \log \text{tw}(G)))$.*

In other words, Theorem 4 states that there exists a constant C such that for every graph G and integers $a, b \geq 2$, if the treedepth of G is at least $Cab \log a$, then the treewidth of G is at least a or G contains a subcubic tree of treedepth b . Since every subcubic tree of treedepth d contains either a simple path of length $2^{\Omega(\sqrt{d})}$ or a subdivision of a full binary tree of depth $\Omega(\sqrt{d})$ [4], we have the following corollary.

► **Corollary 5.** *Let G be a graph of treewidth $\text{tw}(G)$ and treedepth $\text{td}(G)$. Then for some*

$$h = \Omega\left(\sqrt{\text{td}(G)/(\text{tw}(G) \log \text{tw}(G))}\right)$$

G contains either a simple path of length 2^h or a subdivision of a full binary tree of depth h .

Consequently, there exists an absolute constant C such that for every integer $k \geq 1$ and a graph G of treedepth at least $Ck^3 \log k$, either

- *G has treewidth at least k ,*
- *G contains a subdivision of a full binary tree of depth k as a subgraph, or*
- *G contains a path of length 2^k .*

In other words, Corollary 5 improves the bound $k^5 \log^2 k$ of Kawarabayashi and Rossman [4] to $k^3 \log k$. We remark here that there are subcubic trees of treedepth $\Omega(h^2)$ that contains neither a path of length 2^h nor a subdivision of a full binary tree of depth h ,² and thus the quadratic loss between the statements of Theorem 4 and Corollary 5 is necessary.

Inside the proof of Theorem 4 we make use of the following lemma that may be of independent interest. This lemma is the main technical improvement upon the work of Kawarabayashi and Rossman [4].

► **Lemma 6.** *Every tree of treedepth d contains a subcubic subtree of treedepth at least $\frac{\log((1+\sqrt{5})/2)}{\log(3)}d$.*

Furthermore, such a subtree can be found in polynomial time.

The techniques developed to prove Theorem 4 have some implications on the approximability of treedepth. We improve upon Lemma 2 as follows.

► **Theorem 7.** *Given a graph G , one can in polynomial time compute a treedepth decomposition of G of width $\mathcal{O}(\text{td}(G) \cdot \text{tw}(G) \log^{3/2} \text{tw}(G))$.*

² It is straightforward to deduce such an example from the proof of [4]. We provide such an example in Section 6.

34:4 Improved Bounds for the Excluded-Minor Approximation of Treedepth

The result of Kawarabayashi and Rossman [4] has been also an important ingredient in the study of *linear colorings* [5]. A coloring $\alpha : V(G) \rightarrow \mathbb{Z}$ of a graph G is a *linear coloring* if for every (not necessarily induced) path P in G there exists a vertex $v \in V(P)$ of unique color $\alpha(v)$ on P . Clearly, each centered coloring is a linear coloring, but the minimum number of colors needed for a linear coloring can be much smaller than the treedepth of a graph. Kun et al. [5] provided a polynomial relation between the treedepth and the minimum number of colors in a linear coloring; by replacing their usage of [4] by our result (and using an improved bound for the excluded grid theorem [1]) we obtain an improved bound.

► **Theorem 8.** *There exists a polynomial p such that for every integer k and graph G , if the treedepth of G is at least $k^{19}p(\log k)$, then every linear coloring of G requires at least k colors.*

The previous bound of [5] is $k^{190}p(\log k)$.

After proving Lemma 6 in Section 2, we prove Theorem 4 in Section 3. Theorem 7 is proven in Section 3 while Theorem 8 is proven in Section 5. The symbol \log_p stands for base- p logarithm and \log stands for \log_2 . We denote $\varphi = \frac{1+\sqrt{5}}{2}$; note that φ is chosen in a way so that $\varphi^2 = \varphi + 1$ and $\varphi > 1$.

2 Subcubic subtrees of trees of large treedepth

This section focuses on proving Lemma 6.

Schäffer [11] proved that there is a linear time algorithm for finding a vertex ranking with minimum number of colors of a tree T . We follow [5] for a good description of its properties.

In original Schäffer's algorithm ranks are starting from 1, however for the ease of exposition let us assume that ranks are starting from 0. That is, the algorithm constructs a vertex ranking $\alpha : V(T) \rightarrow \{0, 1, 2, \dots\}$ trying to minimize the maximum value attained by α . Assume that T is rooted in an arbitrary vertex and for every $v \in V(T)$ let T_v be the subtree rooted at v .

Of central importance to Schäffer's algorithm are what we will refer to as *rank lists*. For a rooted tree T , the rank list $L(T)$ for vertex ranking α consists of these ranks i for which there exists a path P starting from the root and ending in a vertex v with $\alpha(v) = i$ such that for every $u \in V(P) \setminus \{v\}$ we have $\alpha(u) < \alpha(v)$, that is, v is the unique vertex of maximum rank on P . More formally:

► **Definition 9.** *For a vertex ranking α of tree T , the rank list of T , denoted $L(T)$, can be defined recursively as $L(T) = L(T \setminus T_v) \cup \{\alpha(v)\}$ where v is the vertex of maximum rank in T .*

Schäffer's algorithm arbitrarily roots T and builds the ranking from the leaves to the root of T , computing the rank of each vertex from the rank lists of each of its children. For brevity, we denote $L(v) = L(T_v)$ for every v in T .

► **Proposition 10.** *Let α be a vertex ranking of T produced by Schäffer's algorithm and let $v \in T$ be a vertex with children u_1, \dots, u_l . If x is the largest integer appearing on rank lists of at least two children of v (or -1 if all such rank lists are pairwise disjoint) then $\alpha(v)$ is the smallest integer satisfying $\alpha(v) > x$ and $\alpha(v) \notin \bigcup_{i=1}^l L(u_i)$.*

For a node $v \in V(T)$, and vertex ranking α , the following potential is pivotal to the analysis of Schäffer's algorithm. Let $l_0 > l_1 > \dots > l_{|L(v)|-1}$ be the elements of $L(v)$ sorted in decreasing order.

$$\zeta(v) = \sum_{r \in L(v)} 3^r = \sum_{i=0}^{|L(v)|-1} 3^{l_i}.$$

When we write $\zeta(T)$ for some tree T we refer to $\zeta(s)$ where s is a root of T . For our purposes, we will also use a skewed version of potential function with a different base

$$\sigma(v) = \sum_{i=0}^{|L(v)|-1} \varphi^{l_i-i},$$

where again $l_0 > l_1 > \dots > l_{|L(v)|-1}$ are elements of $L(v)$ sorted in decreasing order. Throughout this section, when focusing on one node $v \in V(T)$, we use notation that l_i is i -th element of set $L(v)$ when sorted in decreasing order and when 0-based indexed.

Let us start with proving following two bounds that estimate $\text{td}(T)$ in terms of $\zeta(T)$ and $\sigma(T)$.

▷ **Claim 11.** $\log_{\varphi}(\sigma(T)) \geq \text{td}(T) - 1$.

Proof. We know that $L(T)$ is nonempty and its biggest element is equal to $\text{td}(T) - 1$ (we need to subtract one because we use nonnegative numbers as ranks, not positive). Therefore we have

$$\sigma(T) = \sum_{i=0}^{|L(T)|-1} \varphi^{l_i-i} \geq \varphi^{l_0} = \varphi^{\text{td}(T)-1}.$$

Hence, $\log_{\varphi}(\sigma(T)) \geq \text{td}(T) - 1$, as desired. ◁

▷ **Claim 12.** $\log_3(\zeta(T)) + \log_3(2) < \text{td}(T)$.

Proof. We have that

$$\begin{aligned} \zeta(T) &= \sum_{r \in L(v)} 3^r \leq \sum_{r=0}^{\text{td}(T)-1} 3^r = \frac{3^{\text{td}(T)} - 1}{2}, \\ 2\zeta(T) &\leq 3^{\text{td}(T)} - 1 < 3^{\text{td}(T)} \end{aligned}$$

$\log_3(2) + \log_3(\zeta(T)) < \text{td}(T)$. ◁

We are ready to prove Lemma 6. Given tree T we want to produce a subcubic (i.e., maximum degree at most 3) tree S which is a subtree of T and that fulfills $\text{td}(S) > \text{td}(T) \log_3(\varphi)$.

Let us start our algorithm by arbitrary rooting T and computing rank lists using Schäffer's algorithm. Then for every vertex $v \in T$ we define $C(v)$ as a set of two children of v that have the biggest value of ζ in case v has at least two children, or all children otherwise. Let us now define forest F whose vertex set is the same as vertex set of T where for every v we put edges between v and all elements of $C(v)$. Clearly this is a forest consisting of subcubic trees which are subtrees of T (where *subtree* is understood as subgraph, not necessarily as some vertex t along with all its descendants in a rooted tree). Let S be a tree of this forest containing root of T . We claim that S is that subcubic subtree of T we are looking for. Note that computing F and thus S can be trivially done in polynomial time. Hence, we are left with proving that $\text{td}(S) > \text{td}(T) \log_3(\varphi)$.

34:6 Improved Bounds for the Excluded-Minor Approximation of Treedepth

Let us root every tree of F in a vertex that was closest to root of T in T . Then compute rank lists for these trees using Schäffer's algorithm. So now, for every vertex we have two rank lists, one for T and one for F . Let us now denote these second ranklists as $\tilde{L}(v)$ for $v \in V(T)$ and let us define function $\tilde{\zeta}$ which will be similar potential function as ζ , but operating on rank lists $\tilde{L}(v)$ instead of $L(v)$. Following claim will be crucial.

▷ **Claim 13.** For every $v \in V(T)$ it holds that $\tilde{\zeta}(v) \geq \sigma(v)$.

We first verify that Claim 13 implies Lemma 6.

Proof of Lemma 6. Using also Claims 12 and 11 we infer that

$$\begin{aligned}
 \text{td}(S) &> \log_3(\tilde{\zeta}(S)) + \log_3(2) && \text{by Claim 12 for } S \\
 &\geq \log_3(\sigma(T)) + \log_3(2) && \text{by Claim 13} \\
 &= \log_\varphi(\sigma(T)) \cdot \log_3(\varphi) + \log_3(2) && \text{logarithm base change} \\
 &\geq (\text{td}(T) - 1) \cdot \log_3(\varphi) + \log_3(2) && \text{by Claim 11 for } T \\
 &= \text{td}(T) \cdot \log_3(\varphi) - \log_3(\varphi) + \log_3(2) > \text{td}(T) \cdot \log_3(\varphi). && \blacktriangleleft
 \end{aligned}$$

Thus it remains to prove Claim 13. To this end, we prove two auxiliary inequalities.

▷ **Claim 14.** For every $v \in V(T)$ it holds that $\tilde{\zeta}(v) \geq 1 + \sum_{s \in C(v)} \tilde{\zeta}(s)$

Proof. We express every $\tilde{\zeta}(x)$ for $x \in \{v\} \cup C(v)$ as a sum of powers of 3 and count how many times each power occurs on both sides of this claimed inequality. Consider a summand 3^c . If $c > \alpha(v)$ then, by the choice of $\alpha(v)$, 3^c appears at most once on the right side and if it appears there, then it appears on the left side as well, so contributions of summands of form 3^c for $c > \alpha(v)$ to both sides are equal. The summand $3^{\alpha(v)}$ appears once on the left side and does not appear on the right side. For $c < \alpha(v)$, the summands of form 3^c appear at most twice in $\sum_{s \in C(v)} \tilde{\zeta}(s)$, so their contribution to right side can be bounded from above by $\sum_{c=0}^{\alpha(v)-1} 2 \cdot 3^c = 3^{\alpha(v)} - 1$, so in fact $3^{\alpha(v)}$ from the left side contributes at least as much as remaining summands from the right side. This finishes the proof of the claim. ◀

▷ **Claim 15.** For every $v \in V(T)$ it holds that $\sigma(v) \leq 1 + \sum_{s \in C(v)} \sigma(s)$

Proof. Recall that by the definition $C(v)$ is a set of two children of v in T with the biggest values of ζ or a set of all children of v in case it has less than two of them. Observe that having bigger value of $\zeta(v)$ is another way of expressing having the set $L(v)$ bigger lexicographically when sorted in decreasing order.

If v is a leaf then $C(v)$ is empty and $\sigma(v) = 1$, so the inequality is obvious. Henceforth we focus on a vertex v that is not a leaf. In our proof following equation will come handy:

$$\varphi = \sum_{i=0}^{\infty} \varphi^{-2i}$$

It holds since $\sum_{i=0}^{\infty} \varphi^{-2i} = \frac{1}{1-\varphi^{-2}} = \frac{\varphi^2}{\varphi^2-1} = \frac{\varphi^2}{\varphi} = \varphi$.

Let us now analyze $L(v)$. It consists of some prefix P of values that appeared exactly once in children of v , then $\alpha(v)$ and then nothing (when enumerated from the biggest to the smallest). Let us now denote by A_i intersection of $L(u_i)$ and P , where u_i is i -th child of v when sorted in nonincreasing order by their values $\zeta(u_i)$ (1-based). We distinguish two cases:

Case 1: A_2 is nonempty. If A_2 is nonempty then in particular it means that v has at least two children. Let us denote the biggest element of $L(u_2)$ by d . We have that $d \in P$, but d is not the biggest element of P . Its contribution to $\sigma(u_2)$ is φ^d , however its contribution to $\sigma(v)$ is at most φ^{d-1} (because of the skew and since d is not the biggest element of P). Contribution to $\sigma(v)$ of elements smaller than d can be bounded from above by $\varphi^{d-3} + \varphi^{d-5} + \dots$. We know that $d = l_j$ for some j , where $j \geq 1$ and $L(v)$ consists of elements $l_0 > l_1 > \dots > l_{|L(v)-1|}$. We have that $l_k \in L(u_1)$ for $k < j$ and that $l_j \geq l_i + (i - j)$ for $i \geq j$, so $l_i - i \leq l_j - j - 2(i - j) = d - j - 2(i - j)$. We can deduce that

$$\begin{aligned} \sigma(v) &= \sum_{i=0}^{|L(v)|-1} \varphi^{l_i-i} = \sum_{i=0}^{j-1} \varphi^{l_i-i} + \sum_{i=j}^{|L(v)|-1} \varphi^{l_i-i} \leq \sigma(u_1) + \sum_{i=j}^{|L(v)|-1} \varphi^{d-j-2(i-j)} \\ &\leq \sigma(u_1) + \varphi^{d-j} \sum_{i=j}^{\infty} \varphi^{-2(i-j)} = \sigma(u_1) + \varphi^{d-j} \sum_{i=0}^{\infty} \varphi^{-2i} = \sigma(u_1) + \varphi^{d-j+1} \\ &\leq \sigma(u_1) + \varphi^d \leq \sigma(u_1) + \sigma(u_2) < 1 + \sigma(u_1) + \sigma(u_2), \end{aligned}$$

which is what we wanted to prove.

Case 2: A_2 is empty. Let us now introduce a few variables:

- d - the biggest integer number smaller than $\alpha(v)$ that is not an element of $L(u_1)$. We know that elements from $d + 1$ to $\alpha(v) - 1$ belong to $L(u_1)$.
- k - shorthand for number of these elements (which is equal to $\alpha(v) - 1 - d$). k can be zero, but cannot be negative.
- g - the number of elements of $L(v)$ that are bigger than $\alpha(v)$.

Then from the definition of $\alpha(v)$ either

- $d = -1$; or
- v has at least two children and $L(u_2)$ contains a number that is at least d .

Because of that we have $1 + \sum_{s \in C(v)} \sigma(s) \geq \sigma(u_1) + \varphi^d$. We know that $\sum_{s \in C(v)}$ is either $\sigma(u_1)$ or $\sigma(u_1) + \sigma(u_2)$, depending on whether v has only one child or more. If $d = -1$ then $1 \geq \varphi^d$ and stated inequality holds. If $d \neq -1$ then u_2 exists and $\sigma(u_2) \geq \varphi^d$.

Note that either $k > 0$ or $g > 0$, because if $k = g = 0$ then $d = \alpha(v) - 1$ and $L(u_1)$ cannot contain elements bigger than $\alpha(v)$ (because $g = 0$), cannot contain $\alpha(v)$ (from the definition of $\alpha(v)$) and cannot contain $\alpha(v) - 1$ (since $d = \alpha(v) - 1$), so its biggest element is at most $d - 1$. If $d = -1$ then it means that v is a leaf, but we already assumed it is not one. However, if v has at least two children and $L(u_2)$ contains a number that is at least d , then it contradicts the assumption that $\zeta(u_1) \geq \zeta(u_2)$. So indeed it holds that $k > 0$ or $g > 0$ and therefore $k + g \geq 1$.

We have that

$$\sigma(v) - \sigma(u_1) \leq \varphi^{\alpha(v)-g} - (\varphi^{\alpha(v)-g-1} + \varphi^{\alpha(v)-g-3} + \dots + \varphi^{\alpha(v)-g-2k+1}),$$

which is because summands coming from numbers bigger than $\alpha(v)$ in $L(v)$ and $L(u_1)$ cancel out (A_2 is empty, so all elements of $L(v)$ different than $\alpha(v)$ come from $L(u_1)$) and new rank $\alpha(v)$ contributes $\varphi^{\alpha(v)-g}$ to $\sigma(v)$ whereas $L(u_1)$ contains numbers from $d + 1$ up to $\alpha(v) - 1$ and their contribution to $\sigma(u_1)$ is $\varphi^{\alpha(v)-g-1} + \varphi^{\alpha(v)-g-3} + \dots + \varphi^{\alpha(v)-g-2k+1}$. We conclude that $\sigma(v) - \sigma(u_1) \leq \varphi^{-g}(\varphi^{\alpha(v)} - (\varphi^{\alpha(v)-1} + \varphi^{\alpha(v)-3} + \dots + \varphi^{\alpha(v)-2k+1}))$.

On the other hand since $\varphi^2 = \varphi + 1$ we have that

$$\begin{aligned} \varphi^{\alpha(v)} &= \varphi^{\alpha(v)-1} + \varphi^{\alpha(v)-2} = \varphi^{\alpha(v)-1} + \varphi^{\alpha(v)-3} + \varphi^{\alpha(v)-4} = \dots = \\ &= (\varphi^{\alpha(v)-1} + \varphi^{\alpha(v)-3} + \dots + \varphi^{\alpha(v)-2k+1}) + \varphi^{\alpha(v)-2k}. \end{aligned}$$

34:8 Improved Bounds for the Excluded-Minor Approximation of Treedepth

Because of that we have

$$\sigma(v) - \sigma(u_1) \leq \varphi^{-g} \cdot \varphi^{\alpha(v) - 2k} = \varphi^{\alpha(v) - 2k - g} = \varphi^{\alpha(v) - (\alpha(v) - 1 - d) - k - g} = \varphi^{d+1 - (k+g)} \leq \varphi^d.$$

From that we conclude that $\sigma(v) \leq \sigma(u_1) + \varphi^d \leq 1 + \sum_{s \in C(v)} \sigma(s)$, what concludes proof of this claim. \triangleleft

Now, having claims 15 and 14 proven, we can wrap our reasoning up. If v is a leaf then $\sigma(v) = \tilde{\zeta}(v) = 1$. If v is not a leaf then we know that $\sigma(v) \leq 1 + \sum_{s \in C(v)} \sigma(s)$ and $\tilde{\zeta}(v) \geq 1 + \sum_{s \in C(v)} \tilde{\zeta}(s)$, so by straightforward induction we get that $\sigma(v) \leq \tilde{\zeta}(v)$ for every $v \in V(T)$, as desired by Claim 13.

3 Proof of Theorem 4

Let G be a nonempty graph and let $r = \text{td}(G)/(\text{tw}(G) + 1)$. Recall that our goal is to show existence of a subcubic tree H being a subgraph of G such that $\text{td}(H)$ is $\Omega(r/\log(\text{tw}(G) + 1))$. Without loss of generality we may assume that G is connected, as otherwise we focus on the connected component of G of maximum treedepth. Also, the statement is trivial for $\text{tw}(G) \leq 1$ (when G is a tree) and when $r \leq 2$, so assume otherwise.

We consider a greedy tree decomposition of G , as defined in [4]. A greedy tree decomposition is a tree decomposition that can be also interpreted as a treedepth decomposition.

Recall that a tree decomposition of a graph G is a pair (T, β) where T is a rooted tree and $\beta : V(T) \rightarrow 2^{V(G)}$ is such that for every $v \in V(G)$ the set $\{t \in V(T) \mid v \in \beta(t)\}$ induces a connected nonempty subtree of T and for every $uv \in E(G)$ there exists $t \in V(T)$ with $u, v \in \beta(t)$. A tree decomposition (T, β) of a graph G is *greedy* if

1. $V(T) = V(G)$,
2. for every $uv \in E(G)$, the nodes u and v in T are in ancestor-descendant relation in T , and
3. for every vertex $u \in V(T)$ and its child v there is some descendant w of v in T such that $uw \in E(G)$.

Lemma 3.6. in [4] states that for every connected graph G there exists a greedy tree decomposition (T, β) of width $\text{tw}(G)$, that is, all bags $\beta(t)$ for $t \in V(T)$ have size bounded by $\text{tw}(G) + 1$. Let (T, β) be such a decomposition of our graph G . By Lemma 1 we get that $\text{td}(T) \geq r$, as otherwise we would be able to construct treedepth decomposition of too low treedepth.

In the remainder of the proof we show the following lemma.

► **Lemma 16.** *Let G be a connected graph, (T, β) be a greedy tree decomposition of G , and let $\tau \geq 2$ be such that $|\beta(t)| \leq \tau$ for every $t \in V(T)$. Then G contains a subcubic tree of treedepth $\Omega(\text{td}(T)/\log \tau)$.*

Theorem 4 follows immediately from Lemma 16 applied to the tree decomposition (T, β) of G . Thus, it remains to prove Lemma 16.

To this end, we first apply Lemma 6 to tree T and obtain a subcubic tree S such that

$$\text{td}(S) \geq r \cdot \log_3(\varphi). \tag{1}$$

Second, we apply the core part of the reasoning of Kawarabayashi and Rossman [4]. The construction of Section 5 of [4] can be encapsulated in the following lemma.

► **Lemma 17** (Section 5 of [4]). *Let (T, β) be a greedy tree decomposition of graph G and let $\tau = \max_{t \in V(T)} |\beta(t)|$. Then for every subcubic subtree S of T there exists a subtree F of G such that $V(S) \subseteq V(F)$ and the maximum degree of F is bounded by $\tau + 2$.*

By application of Lemma 17 to our decomposition (T, β) and subtree S we get a tree F in G , which has large treedepth, as we show in a moment. To this end, we need the following simple bound on treedepth of trees.

► **Lemma 18.** *For every tree H with maximum degree bounded by $d \geq 2$ it holds that*

$$\log_d |V(H)| \leq \text{td}(T) \leq 1 + \log_2 |V(H)|.$$

Proof. We use the following equivalent recursive definition of treedepth: Treedepth of an empty graph is 0, treedepth of a disconnected graph equals the maximum of treedepth over its connected components, while for nonempty connected graphs G we have $\text{td}(G) = 1 + \min_{v \in V(G)} \text{td}(G - v)$.

For the lower bound, for $k \geq 1$ let $f_d(k)$ be the maximum possible number of vertices of a tree of maximum degree at most d and treedepth at most k . Clearly, $f_d(1) = 1$. Since removing a single vertex from a tree of maximum degree at most d results in at most d connected components, we have that

$$f_d(k+1) \leq 1 + d \cdot f_d(k).$$

Consequently, we obtain by induction that

$$f_d(k) \leq d^k - 1.$$

This proves the lower bound. For the upper bound, note that in every tree T there exists a vertex $v \in V(T)$ such that every connected component of $T - \{v\}$ has at most $|V(T)|/2$ vertices. Consequently, if we define $g(n)$ to be the maximum possible treedepth of an n -vertex tree, then $g(1) = 1$ and we have that

$$g(n) \leq 1 + g(\lfloor n/2 \rfloor).$$

This proves the upper bound. ◀

By (1) and Lemma 18 we get that $|V(S)| \geq 2^{r \cdot \log_3(\varphi) - 1}$. This implies that also

$$|V(F)| \geq 2^{r \cdot \log_3(\varphi) - 1}. \quad (2)$$

As S is subcubic, by Lemma 17 we know that the maximum degree of F is bounded by $\text{tw}(G) + 3$. Therefore Lemma 18 and (2) jointly imply that

$$\text{td}(F) \geq \log_{(\text{tw}(G)+3)} 2^{r \cdot \log_3(\varphi) - 1} \geq \frac{r \cdot \log_3(\varphi) - 1}{\log(\text{tw}(G) + 3)} \geq \frac{\log_3(\varphi)}{20} \cdot r / \log(\text{tw}(G) + 1). \quad (3)$$

Here, the last inequality follows from the assumptions $r > 2$ and $\text{tw}(G) \geq 2$; note that the constant 20 is sufficiently large constant for the estimations to work.

As tree F is not necessarily subcubic, we apply one more time Lemma 6 and get a subcubic subtree H of F such that

$$\text{td}(H) \geq \text{td}(F) \cdot \log_3(\varphi) \geq \frac{\log_3(\varphi)^2}{20} \cdot r / \log(\text{tw}(G) + 1), \quad (4)$$

which finishes the proof of Lemma 16 and of Theorem 4.

4 Proof of Theorem 7

Proof of Theorem 7. Without loss of generality we can assume that the input graph G is connected. As in the proof of Lemma 2, we apply the polynomial-time approximation algorithm for treewidth [3], to compute a tree decomposition (T_0, β_0) of G with $\mathcal{O}(n)$ nodes of T_0 and $|\beta(t)| \leq \tau$ for every $t \in V(T_0)$ and some $\tau = \mathcal{O}(\text{tw}(G)\sqrt{\log \text{tw}(G)})$. As discussed in [4], one can in polynomial time turn (T_0, β_0) into a greedy tree decomposition (T, β) of G without increasing the maximum size of a bag, that is, still $|\beta(t)| \leq \tau$ for every $t \in V(T)$. We apply Lemma 1 to (T, β) , returning a treedepth decomposition of G of width at most $\tau \cdot \text{td}(T) = \mathcal{O}(\text{td}(T)\text{tw}(G)\sqrt{\log \text{tw}(G)})$.

It remains to bound $\text{td}(T)$. Lemma 16 asserts that G contains a subcubic tree H of treedepth $\Omega(\text{td}(T)/\log \tau)$. Therefore $\text{td}(T) = \mathcal{O}(\text{td}(H) \log \tau) = \mathcal{O}(\text{td}(G) \log \text{tw}(G))$ and thus the width of the computed treedepth decomposition is $\mathcal{O}(\text{td}(G)\text{tw}(G) \log^{3/2} \text{tw}(G))$. This finishes the proof of Theorem 7. \blacktriangleleft

5 Proof of Theorem 8

Here we show how to assemble the proof of Theorem 8 from Theorem 4, a number of intermediate results of [5], and an improved excluded grid theorem due to Chuzhoy and Tan [1]:

► **Theorem 19** ([1]). *There exists a polynomial p_{GMT} such that for every integer k if a graph G has treewidth at least $k^9 p_{\text{GMT}}(\log k)$ then G contains a $k \times k$ grid as a minor.*

The following two results were proven in [5].

► **Lemma 20** ([5]). *If a graph G contains a $k \times k$ grid as a minor, then every linear coloring of G requires $\Omega(\sqrt{k})$ colors.*

► **Lemma 21** ([5]). *If G is a tree of treedepth d and maximum degree Δ , then every linear coloring of G requires at least $d/\log_2(\Delta)$ colors.*

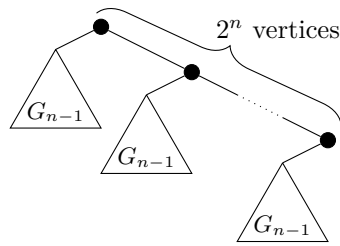
Recall that Theorem 4 asserts that there exists a constant C such that for every graph G and integers $a, b \geq 2$, if the treedepth of G is at least $Cab \log a$, then either the treewidth of G is at least a or G contains a subcubic tree of treedepth at least b . Applying this theorem to $a = \theta(k^2)$ and $b = k \log_2(3)$, one obtains that if the treedepth of G is $\Omega(k^{19} p_{\text{GMT}}(\log k) \log k)$, then G contains either a $\theta(k^2) \times \theta(k^2)$ grid minor or a subcubic tree of treedepth at least $k \log_2(3)$. In the first outcome, Lemma 20 gives the desired number of colors of a linear coloring, while in the second outcome the same result is obtained from Lemma 21. This concludes the proof of Theorem 8.

6 An example of a tree with treedepth quadratic in the height of the binary tree or logarithm of a length of a path

In this section we provide a construction of a family of trees $(G_n)_{n \geq 1}$ such that

1. The tree G_n does not contain a path with 2^{n+2} vertices.
2. The tree G_n does not contain a subdivision of a full binary tree of depth $n + 2$.
3. The treedepth of G_n is at least $\binom{n+1}{2}$.

We will consider each tree G_n as a rooted tree. The tree G_1 consists of a single vertex. For $n \geq 2$, the tree G_n is defined recursively as follows. We take a path P_n with 2^n vertices and for each $v \in V(P_n)$ we create a copy G_n^v of G_{n-1} and attach its root to v . We root G_n in one of the endpoints of P_n ; see Figure 1. We now proceed with the proof of the properties of G_n .



■ **Figure 1** Construction of G_n .

Since every path in G_n is contained in at most two root-to-leaf paths (not necessarily edge-disjoint), to show Property (1) it suffices to show the following.

► **Lemma 22.** *Every root-to-leaf path in G_n contains less than 2^{n+1} vertices.*

Proof. We prove the statement by induction on n . For $n = 1$ the statement is straightforward. For the inductive step, observe that every root-to-leaf path in G_n consists of a subpath of P_n (which has 2^n vertices) and a root-to-leaf path in one of the copies C_n^v of G_{n-1} (which has less than 2^n vertices by the inductive assumption). ◀

We say that a subtree H of G_n that is a subdivision of a full binary tree of height $h \geq 1$ is *aligned* if $h = 1$ or $h \geq 2$ and the closest to the root vertex of H is of degree 2 in H and its deletion breaks H into two subtrees containing a subdivision of a full binary tree of height $h - 1$. In other words, an aligned subtree has the same ancestor-descendant relation as the tree G_n . Observe that any subtree H_0 of G_n that is a subdivision of a full binary tree of height $h \geq 2$ contains a subtree that is an aligned subdivision of a full binary tree of height $h - 1$. Therefore, to prove Property (2), it suffices to show the following.

► **Lemma 23.** *G_n does not contain an aligned subdivision of a full binary tree of height $n + 1$.*

Proof. We prove the claim by induction on n . It is straightforward for $n = 1$. For $n \geq 2$, let H be such an aligned subtree of G_n and let w be the closest to the root of G_n vertex of H . If $w \in V(C_n^v)$ for some $v \in V(P_n)$, then H is completely contained in C_n^v , which is a copy of G_{n-1} . Otherwise, $w \in V(P_n)$ and thus one of the components of $H - \{w\}$ lies in C_n^w . However, this component contains an aligned subdivision of a full binary tree of height n . In both cases, we obtain a contradiction with the inductive assumption. ◀

We are left with the treedepth lower bound of Property (3). To this end, we consider the following families of trees. For integers $a, b \geq 1$, the family $\mathcal{G}_{a,b}$ contains all trees H that are constructed from a path P_H with at least 2^a vertices by attaching, for every $v \in V(P_H)$, a tree T_v of treedepth at least b by an edge to v . We show the following.

► **Lemma 24.** *For every $H \in \mathcal{G}_{a,b}$ we have $\text{td}(H) \geq a + b$.*

Proof. We prove the lemma by induction on a . For $a = 1$ we have $\text{td}(H) \geq a + 1$ and H contains two vertex-disjoint subtrees of treedepth at least b each. Assume then $a > 1$ and $H \in \mathcal{G}_{a,b}$. Then for every $v \in V(H)$, $H - v$ contains a connected component that contains a subtree belonging to $\mathcal{G}_{a-1,b}$. This finishes the proof. ◀

We show Property (3) by induction on n . Clearly, $\text{td}(G_1) = 1 = \binom{1+1}{2}$. Consider $n \geq 2$. Since the treedepth of G_{n-1} is at least $\binom{n}{2}$, we have that $G_n \in \mathcal{G}_{n, \binom{n}{2}}$. By Lemma 24, we have that

$$\text{td}(G_n) \geq n + \binom{n}{2} = \binom{n+1}{2}.$$

This finishes the proof of Property (3).

7 Conclusions

We have provided improved bounds for the excluded minor approximation of treedepth of Kawarabayashi and Rossman [4]. Our main result, Theorem 4, is close to being optimal in the following sense: as witnessed by the family of trees, if one considers the measure $r := \text{td}(G)/\text{tw}(G)$, one cannot hope to find a tree in G of treedepth larger than r . We pose getting rid of the $\log(\text{tw}(G) + 1)$ factor in Theorem 4 as an open problem. Improving the $Ck^3 \log k$ bound of Corollary 5 to $Ck^{3-\varepsilon}$ for some $\varepsilon > 0$ seems much more challenging.

Our main result can be applied to a polynomial-time treedepth approximation algorithm, improving upon state-of-the-art tradeoff trick. As a second open problem, we ask for a polynomial-time or single-exponential in treedepth parameterized algorithm for constant or polylogarithmic approximation of treedepth.

References

- 1 Julia Chuzhoy and Zihan Tan. Towards Tight(er) Bounds for the Excluded Grid Theorem. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1445–1464. SIAM, 2019. doi:10.1137/1.9781611975482.88.
- 2 Zdenek Dvorak, Archontia C. Giannopoulou, and Dimitrios M. Thilikos. Forbidden graphs for tree-depth. *Eur. J. Comb.*, 33(5):969–979, 2012. doi:10.1016/j.ejc.2011.09.014.
- 3 Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved Approximation Algorithms for Minimum Weight Vertex Separators. *SIAM J. Comput.*, 38(2):629–657, 2008. doi:10.1137/05064299X.
- 4 Ken-ichi Kawarabayashi and Benjamin Rossman. A Polynomial Excluded-Minor Approximation of Treedepth. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 234–246. SIAM, 2018. doi:10.1137/1.9781611975031.17.
- 5 Jeremy Kun, Michael P. O’Brien, and Blair D. Sullivan. Treedepth Bounds in Linear Colorings. *CoRR*, abs/1802.09665, 2018. arXiv:1802.09665.
- 6 Jaroslav Nesetril and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006. doi:10.1016/j.ejc.2005.01.010.
- 7 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.
- 8 Jaroslav Nesetril and Patrice Ossona de Mendez. On Low Tree-Depth Decompositions. *Graphs and Combinatorics*, 31(6):1941–1963, 2015. doi:10.1007/s00373-015-1569-7.
- 9 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A Faster Parameterized Algorithm for Treedepth. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014. doi:10.1007/978-3-662-43948-7_77.

- 10 Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.
- 11 Alejandro A. Schäffer. Optimal Node Ranking of Trees in Linear Time. *Inf. Process. Lett.*, 33(2):91–96, 1989. doi:10.1016/0020-0190(89)90161-0.

Building a Nest by an Automaton

Jurek Czyzowicz

Département d'informatique, Université du Québec en Outaouais, Canada
jurek@uqo.ca

Dariusz Dereniowski 

Faculty of Electronics, Telecommunications and Informatics,
Gdańsk University of Technology, Poland
deren@eti.pg.edu.pl

Andrzej Pelc

Département d'informatique, Université du Québec en Outaouais, Canada
pelc@uqo.ca

Abstract

A robot modeled as a deterministic finite automaton has to build a structure from material available to it. The robot navigates in the infinite oriented grid $\mathbb{Z} \times \mathbb{Z}$. Some cells of the grid are full (contain a brick) and others are empty. The subgraph of the grid induced by full cells, called the *field*, is initially connected. The (Manhattan) distance between the farthest cells of the field is called its *span*. The robot starts at a full cell. It can carry at most one brick at a time. At each step it can pick a brick from a full cell, move to an adjacent cell and drop a brick at an empty cell. The aim of the robot is to construct the most compact possible structure composed of all bricks, i.e., a *nest*. That is, the robot has to move all bricks in such a way that the span of the resulting field be the smallest.

Our main result is the design of a deterministic finite automaton that accomplishes this task and subsequently stops, for every initially connected field, in time $O(sz)$, where s is the span of the initial field and z is the number of bricks. We show that this complexity is optimal.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases finite automaton, plane, grid, construction task, brick, mobile agent, robot

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.35

Related Version A full version of the paper is available at <https://arxiv.org/abs/1904.10850>.

Funding *Jurek Czyzowicz*: Supported in part by NSERC discovery grant.

Dariusz Dereniowski: Partially supported by National Science Centre (Poland) grant number 2015/17/B/ST6/01887.

Andrzej Pelc: Supported in part by NSERC discovery grant 2018-03899 and by the Research Chair in Distributed Computing of the Université du Québec en Outaouais.

1 Introduction

The problem

A mobile agent (robot) modeled as a deterministic finite automaton has to build a structure from material available to it. The robot navigates in the infinite oriented grid $\mathbb{Z} \times \mathbb{Z}$ represented as the set of unit square cells in the two-dimensional plane, with all cell sides vertical or horizontal. The robot has a compass enabling it to move from a currently occupied cell to one of the four cells (to the North, East, South, West) adjacent to it. Some cells of the grid contain a brick, i.e., are *full*, other cells are *empty*. The subgraph of the grid induced by full cells, called the *field*, is initially connected. The (Manhattan) distance between the farthest cells of the field is called its *span*. Notice that the span of any current field may be much smaller than its diameter as a subgraph of the grid. In fact, this diameter may be sometimes undefined, if the field becomes disconnected. The robot starts at a full cell. It



© Jurek Czyzowicz, Dariusz Dereniowski, and Andrzej Pelc;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 35; pp. 35:1–35:14



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

35:2 Building a Nest by an Automaton

can carry at most one brick at a time. At each step, the robot can pick up a brick from the currently occupied full cell (if it does not carry any brick at this time), moves to an adjacent cell, and can drop a brick at the currently occupied empty cell (if it carries a brick). The robot has no a priori knowledge of the initial field, of its span or of the number of bricks.

The aim of the robot is to construct the most compact possible structure composed of all bricks, i.e., a *nest*. That is, the robot has to move all bricks in such a way that the span of the resulting field be the smallest. The above task has many real applications. In the natural world, animals use material scattered in a territory (pieces of wood, small branches, leaves) to build a nest, and minimizing the span is important to better protect it. A mobile robot may be used to clean a territory littered by hazardous material, in which case minimizing the span of the resulting placement of contaminated pieces facilitates subsequent decontamination. A more mundane example is the everyday task of sweeping the floor, whose aim is to gather all trash in a small space and then get rid of it.

Our results

Our main result is the design of a deterministic finite automaton that accomplishes the task of building a nest and subsequently stops, for every initially connected field, in time $O(sz)$, where s is the span of the initial field and z is the number of bricks. The time is defined as the number of moves of the robot. We show that this complexity is optimal.

The essence of our nest building algorithm is to instruct the robot to make a series of trips to get consecutive bricks, one at a time, and carry them to some designated compact area. This approach ensures the optimal complexity. (In order to show where the problem is, we also sketch a much simpler algorithm that uses another approach but has significantly larger complexity). There are two major difficulties to carry out this plan. The first is that the span of the initial field may be much larger than the memory of the robot, and hence the robot that already put several bricks in a compact area and goes for the next brick cannot remember the way back to the area where it started building. Thus we need to prepare the way, so that the robot can recognize the backtrack path locally at each decision point. The second problem is that, while we temporarily disconnect the field during the execution of the algorithm, special care has to be taken so that the connected components of intermediary fields be close to each other, to prevent the robot from getting lost in large empty spaces.

To the best of our knowledge, the task of constructing structures from available material using an automaton, has never been studied before in the algorithmic setting. It is interesting to compare this task to that of exploration of mazes by automata, that is a classic topic with over 50 years of history (see the section “Related work”). It follows from the result of Budach [9] (translated to our terminology) that if an automaton can only navigate in the field without moving bricks then it cannot explore all connected fields, even without the stop requirement, i.e., it cannot even see all bricks. By contrast, it follows from our result that the ability of moving bricks enables the automaton not only to see all bricks but to build a potentially useful structure using all of them and stop, and to accomplish all of that with optimal complexity.

The model

We consider the infinite oriented grid $\mathbb{Z} \times \mathbb{Z}$ represented as the set of unit square cells tiling the two-dimensional plane, with all cell sides vertical or horizontal. Each cell has 4 adjacent cells, North, East, South and West of it. Some cells of the grid contain a brick, i.e., are *full*, other cells are *empty*. The subgraph of the grid induced by full cells is initially connected.

At each step of the algorithm this subgraph can change, due to the actions of the robot, described below. At each step, the subgraph induced by the full cells is called the current *field*. Any maximal connected subgraph of the current field is called a *component*. Throughout the paper, the *distance* between two cells (x, y) and (x', y') of the grid is the Manhattan distance between them, i.e., $|x - x'| + |y - y'|$. The number of cells of a field is called its *size*, and the distance between two farthest cells of a field is called its *span*. A nest of size z is a field that has the minimum span among all fields of size z .

We are given a mobile entity (robot) starting in some cell of the initial field and traveling in the grid. The robot has a priori no knowledge of the field, of its size or of its span. The robot is formalized as a finite deterministic Mealy automaton $\mathcal{R} = (X, Y, \S, \delta, \lambda, S_0, S_f)$. $X = \{e, f\} \times \{l, h\}$ is the input alphabet, $Y = \{N, E, S, W\} \times \{e, f\} \times \{l, h\}$ is the output alphabet. \S is a finite set of states with two special states: S_0 called initial and S_f called final. $\delta : \S \times X \rightarrow \S$ is the state transition function, and $\lambda : \S \times X \rightarrow Y$ is the output function.

The meaning of the input and output symbols is the following. At each step of its functioning, the robot is at some cell of the grid and has some weight: it is either light, denoted by l (does not carry a brick) or heavy, denoted by h (carries a brick). Moreover, the current cell is either empty, denoted by e or full, denoted by f . The input $x \in \{e, f\} \times \{l, h\}$ gives the automaton information about these facts. The robot is in some state S . Given this state and the input x , the robot outputs the symbol $\lambda(x, S) \in \{N, E, S, W\} \times \{e, f\} \times \{l, h\}$ with the following meaning. The first term indicates the adjacent cell to which the robot moves: North, East, South or West of the current cell. The second term determines whether the robot leaves the current cell empty or full, and the third term indicates whether the robot transits as heavy or as light to the adjacent cell. Since the robot can only either leave the current cell intact and not change its own weight, or pick a brick from a full cell leaving it empty (in the case when the robot was previously light), or drop a brick on an empty cell leaving it full (in the case when the robot was previously heavy), we have the following restrictions on the possible values of the output function λ :

- $\lambda(S, e, l)$ must be (\cdot, e, l)
- $\lambda(S, e, h)$ must be either (\cdot, e, h) or (\cdot, f, l)
- $\lambda(S, f, l)$ must be either (\cdot, f, l) or (\cdot, e, h)
- $\lambda(S, f, h)$ must be (\cdot, f, h)

Seeing the input symbol x and being in a current state S , the robot makes the changes indicated by the output function (it goes to the indicated adjacent cell, possibly changes the filling of the current cell as indicated and possibly changes its own weight as indicated), and transits to state $\delta(x, S)$. The robot starts light in a full cell in the initial state S_0 (hence its initial input symbol is (f, l)) and terminates its action in the final state S_f .

Related work

Problems concerning exploration and navigation performed by mobile agents or robots in an unknown environment have been studied for many years (cf. [7, 21, 26]). The relevant literature can be divided into two parts, according to the environment where the robots operate: it can be either a geometric terrain, possibly with obstacles, or a network modeled as a graph in which the robot moves along edges.

In the geometric context, a closely related problem is that of pattern formation [12, 14, 28]. Robots, modeled as points freely moving in the plane have to arrange themselves to form a pattern given as input. This task has been mostly studied in the context of asynchronous oblivious robots having full visibility of other robots positions.

The graph setting can be further specified in two different ways. In [1, 3, 4, 13, 19] the robot explores strongly connected directed graphs and it can move only in the tail-to-head direction of an edge, not vice-versa. In [2, 5, 9, 15, 16, 17, 25, 27] the explored graph is undirected and the robot can traverse edges in both directions. Graph exploration scenarios can be also classified in another important way. It is either assumed that nodes of the graph have unique labels which the robot can recognize (as in, e.g., [13, 17, 25]), or it is assumed that nodes are anonymous (as in, e.g., [3, 4, 9, 10, 27]). In our case, we work with the infinite anonymous grid, hence it is an undirected anonymous graph scenario. The efficiency measure adopted in papers dealing with graph exploration is either the completion time of this task, measured by the number of edge traversals, (cf., e.g., [25]), or the memory size of the robot, measured either in bits or by the number of states of the finite automaton modeling the robot (cf., e.g., [15, 20, 19]). We are not concerned with minimizing the memory size but we assume that this memory is bounded, i.e., it is constant as a function of the input grid size. However we want to minimize the time of our construction task.

The capability of a robot to explore anonymous undirected graphs has been studied in, e.g., [6, 9, 15, 20, 23, 27]. In particular, it was shown in [27] that no finite automaton can explore all cubic planar graphs (in fact no finite set of finite automata can cooperatively perform this task). Budach [9] proved that a single automaton cannot explore all mazes (that we call connected fields in this paper). Hoffmann [22] proved that one pebble does not help to do it. By contrast, Blum and Kozen [6] showed that this task can be accomplished by two cooperating automata or by a single automaton with two pebbles. The size of port-labeled graphs which cannot be explored by a given robot was investigated in [20].

Recently a lot of attention has been devoted to the problem of searching for a target hidden in the infinite anonymous oriented grid by cooperating agents modeled as either deterministic or probabilistic automata. Such agents are sometimes called ants. It was shown in [18] that 3 randomized or 4 deterministic automata can accomplish this task. Then matching lower bounds were proved: the authors of [11] showed that 2 randomized automata are not enough for target searching in the grid, and the authors of [8] proved that 3 deterministic automata are not enough for this task. Searching for a target in the infinite grid with obstacles was considered in [24].

Our present paper adopts the same model of environment as the above papers, i.e., the infinite anonymous oriented grid. However the task we study is different: instead of searching for a target, the robot has to build some structure from the available material. To the best of our knowledge, such construction tasks performed by automata have never been studied previously in the algorithmic setting.

2 Terminology and preliminaries

In the description and analysis of our algorithm we will categorize full cells. A full cell is said to be a *border cell* if it is adjacent to an empty cell. A full cell that has only one full adjacent cell is called a *leaf*. A full cell is called *special*, if it is either a leaf, or has at least two full cells adjacent to it, sharing a corner.

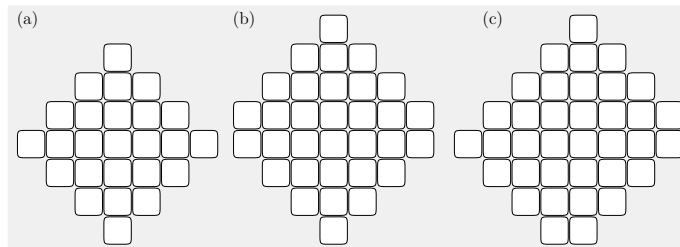
A finite deterministic automaton may remember a constant number of bits by encoding them in its states. We will use this fact to define several simple procedures and simplifications that we use in the sequel. The first simplification is as follows. We formulate the actions of the robot based on the configuration of bricks in its neighborhood. More precisely, at any step, the robot knows whether each cell at distance at most $r = 8$ from its current cell is full or empty. This can be achieved by performing a bounded local exploration with return, after each move of the robot.

We will use the notion of current *orientation* of the robot. At the beginning of its navigation, the robot goes in one of the four cardinal directions. Then its orientation is determined in one of the two ways: either by its last step (North, East, South or West) or by a *turn*: we say that the robot *turns left* (respectively *right*) meaning that it changes its orientation in the appropriate way while remaining at the same cell. Clearly the robot can remember its orientation, using its states. We refer to cardinal directions with respect to this current orientation. Thus, e.g., if the robot is oriented East then we say that its adjacent North (resp. East, South or West) cell is *left* (resp. *in front*, *right*, *back*) of it.

Whenever we say that the robot located at a cell c and not carrying a brick *brings* a brick from a full cell c' to c we mean that the robot moves from c to c' , picks the brick from c' , moves back to c and restores its original orientation. Whenever we say that the robot located at a cell c and carrying a brick *places* it at an empty cell c' we mean that the robot moves from c to c' , drops the brick at c' , moves back to c and restores its original orientation.

Whenever the robot selects a cell according to some condition that is fulfilled by more than one cell, the robot selects the cell that is minimal with respect to the following total order \prec on the set of all cells. For cells $c = (x, y)$ and $c' = (x', y')$, $c \prec c'$ holds if and only if either $y < y'$, or $y = y'$ and $x \leq x'$. We denote by $|S|$ the number of cells in a sequence or a set S of cells.

We define a *disc* of radius $r \geq 0$ with center c to be the set of all cells at distance at most r from c , see Fig. 1. A disc of radius r contains $z_r = 2(1 + 3 + 5 + \dots + (2r - 1)) + (2r + 1) = 2r^2 + 2r + 1$ cells and has span $2r$. A *rough disc* of size z , where $z_r \leq z < z_{r+1}$ is defined as follows. If $z = z_r$, then the rough disc is the disc of radius r . Otherwise, let F be the set of cells not belonging to the disc D of radius r but adjacent to some cell of it. Add to D exactly $z - z_r$ cells belonging to F , starting from the North neighbor of the East-most cell of D and going counterclockwise. If $z_r < z \leq z_r + 2r + 2$ then the rough disc of size z has span $2r + 1$ and if $z_r + 2r + 2 < z < z_{r+1}$ then the rough disc of size z has span $2r + 2$, the same as the disc of radius $r + 1$ that has size z_{r+1} .



■ **Figure 1** (a) disc of size z_r for $r = 3$; (b) a rough disc of size $z_r + 7$ and span $2r + 1$, $r = 3$; (c) a rough disc of size $z_r + 11$ and span $2r + 2$, $r = 3$.

The proofs of the next two propositions are omitted due to space limitation.

► **Proposition 1.** *Any rough disc is a nest.*

The nests built by our automaton will be rough discs. The following proposition shows that the complexity of our nest-building algorithm is optimal, regardless of the relation between the size of the initial field and its span (recall that, by definition, the span must be smaller than the size z and it must be in $\Omega(\sqrt{z})$). Our lower bound on complexity follows from geometric properties of the grid, and hence it holds regardless of the machine that builds the nest, i.e., even if the robot is a Turing machine knowing a priori the initial field.

► **Proposition 2.** *Let $s' < z$ be positive integers such that $s' \in \Omega(\sqrt{z})$. There exists an initial field of size z and span $s \in \Theta(s')$, such that any algorithm that builds a nest starting from this field must use time $\Omega(sz)$.*

3 The algorithm

The robot will move bricks from the original field and build two special components. One of them will be a rough disc that will be gradually extended. The second one will be a one-cell component whose only cell is called the *marker*. The robot will periodically get at large distances from the rough disc being built, and the role of the marker will be to indicate to the robot that it got back in the vicinity of the rough disc. Any component that is different from the rough disc and from the marker will be called a *free component*. During the execution of the entire algorithm, the robot will not ensure that the full cells outside of the rough disc and of the marker form one component – they may form several components – but after adding a new brick to the rough disc these components will be always at a bounded distance, i.e., at distance $O(1)$, from the rough disc that the robot is constructing.

We are now ready to sketch the high-level idea of the algorithm, whose pseudo-code is presented at the end of this section as algorithm Nest. First the robot performs some preliminary actions by establishing the marker and the initial rough disc and by calling procedure Sweep, which together result in constructing the first rough disc D (of size one), placing the marker next to it and ensuring that no full cells other than the marker are at distance at most 7 from D . Then each iteration of the main loop of algorithm Nest performs three actions. First, it executes procedure FindNextBrick that allows the robot to find a brick in a free component \mathcal{C} . This brick must be carefully chosen. For example, greedily picking the closest available brick would soon result in creating large empty spaces between components of the field, in which the robot could get lost. This brick will be later used to extend the rough disc. However, this procedure may lead the robot far from the rough disc and may also disconnect \mathcal{C} into many components. Disconnecting \mathcal{C} is one of the main tools in our construction. It is done by the robot on purpose to allow it to find its way back to the marker and so that it is possible to recover the connectivity of \mathcal{C} on the way back. Such a walk back to the marker is the second action performed in the main loop and described as procedure ReturnToMarker. The third action is done once the robot is back at the marker, and it is given as procedure ExtendRoughDisc. This procedure extends the rough disc, ensuring the property that there are no full cells at a prescribed bounded distance from the rough disc, except the marker. While restoring this property, the robot may again disconnect some components but all of them are at a bounded distance from the rough disc and thus the robot will be able to find them easily. Additionally, it may happen that the cell brought to the rough disc was the last cell of \mathcal{C} . In such a case, as the last part of procedure ExtendRoughDisc, the robot places the marker near another component close to the rough disc, if one exists. This will be the new free component \mathcal{C} in which the robot will find the next brick in the next iteration of the main loop. If no such \mathcal{C} exists, then the robot adds the brick from the marker to the rough disc, thus completing the construction of the nest.

Many of the difficulties described above come from our desire to keep the complexity of the algorithm optimal, i.e., $O(sz)$. If complexity were not an issue, the following much simpler algorithm would be sufficient. The robot first builds a horizontal line at the level of a South-most cell of the initial field, by gradually squeezing down the field, keeping it connected at all times. Then it transforms the line into a nest. The squeezing down can be performed by iteratively repeating the following steps. First, the robot makes sure that it is not on the lowest level (if the line is not yet constructed, this can be done by finding a full cell with a full South neighbor). Then the robot goes to some North-West extremity of the field, i.e., to any full cell that has empty cells to the North and to the West of it. Then it picks the brick from this cell and drops it one level down, ensuring the connectivity. This

can be done by first moving one level down and then iteratively going West until an empty cell is encountered, where the robot drops the brick. When the field is squeezed down to a line, the robot will recognize this and easily transform it into a nest.

This idea potentially requires time $\Theta(z)$ to lower a brick by one level. Since there are z bricks possibly on $\Theta(s)$ levels, the entire time would be $\Theta(sz^2)$ in the worst case, which is suboptimal. Thus we proceed with the detailed description of the optimal algorithm Nest whose high-level idea was described before.

3.1 Moving bricks out of the way

One of the challenges in constructing the rough disc is to have enough room so that, while expanding, it does not merge with the remainder of the field. This is one of the goals of procedure Sweep. Its high-level idea is the following. It ensures an invariant that has to hold whenever the agent goes to retrieve the next brick in order to extend the rough disc. This invariant is that there are no full cells, other than the cell M that is the marker, within a given bounded distance from the current rough disc D . This procedure is called when the robot is at the marker, in two situations. The first one is right before the main loop: in this case the marker, the rough disc (of size one) and its corresponding neighborhood occupy a constant number of cells and hence in this case the robot is able to decide whether a given cell is in $D \cup \{M\}$. The second situation occurs after each extension of the rough disc. In this case, the size of D may be unbounded but a walk around its border (which can be done with stop, using the marker) allows to determine if there is a full cell c within a bounded distance from D , that does not belong to the rough disc itself. Whenever such a full cell c is found, the robot picks the brick from c and searches for an empty cell at distance at least 7 from the rough disc. This searching walk is done in such a way as to ensure the return to the rough disc. At the end of the procedure, the robot places the marker next to some free component. Below is the pseudo-code of procedure Sweep. In this pseudo-code, we use

■ **Procedure Sweep** sweeping away bricks that are close to the rough disc.

```

1  $M :=$  the marker
2 go to the closest cell of the rough disc  $D$ 
3 perform a full counterclockwise traversal of the border of  $D$ , executing the following
  actions after each step:
4   for each full cell  $c \notin D \cup \{M\}$  at distance at most 7 from the robot do
5      $c' :=$  the cell currently occupied by the robot
6     go to  $c$  and pick the brick
7     move in direction away from  $D$  and stop at the first empty cell at distance at
      least 7 from  $D$ 
8     drop the brick and return to  $c'$ 
9 if there exists a free component  $\mathcal{C}$  then
10  pick the brick from marker and place it at distance 3 from the rough disc and at
    distance 4 from  $\mathcal{C}$ , creating a new marker
11 go to the marker

```

the notion of the robot going *in direction away* from the rough disc D . This is the direction which strictly increases the distance between the robot and the rough disc. In the case where there are two such directions, we use priority North, East, South, West.

3.2 Finding the next brick

The high-level idea of procedure FindNextBrick is the following. It may happen that the cells that belong to a free component \mathcal{C} and are close to the marker cannot be rearranged in such a way that the robot be able to obtain a brick that it can then use to extend the rough disc and at the same time keep the connectivity of \mathcal{C} and ensure that \mathcal{C} remains close to the marker. Thus, the robot has to retrieve the needed brick by following a potentially long walk; we will call it a *search walk* and formally define it below. The search walk needs to be carefully chosen to ensure that it ends at a location where it is possible to find the desired brick and so that the robot be able to return to the marker. Moreover, this walk has to be sufficiently short to guarantee the time $O(sz)$ of the algorithm, i.e., the length of each walk must be $O(s)$. We ensure the latter as follows: each search walk \mathcal{W} leads alternately in two non-opposite directions, e.g., North and West. The return is guaranteed by repeatedly performing an action called *switch* while walking along \mathcal{W} , which we formally describe later. Intuitively speaking, the switch eliminates the cells adjacent to \mathcal{W} at which the robot may incorrectly turn on its way back to the marker. The switch potentially disconnects the component but the robot is able to recover the connectivity while backtracking along \mathcal{W} . Finally, we will describe how the desired brick can be found at the end of \mathcal{W} .

3.2.1 Search walks

Suppose that the robot is currently at a full cell and there is a full cell in front of the robot. A *left-free* (respectively *right-free*) segment consists of all full cells that will be visited by the robot that moves without changing its direction until one of the following conditions holds:

- (S1) the robot arrives at a full cell that has an empty cell in front of it and has an empty cell to the left (respectively right) of it; such a segment is called *terminal*,
- (S2) the robot arrives at the first special cell such that the cell to the left (respectively right) of the robot is full.

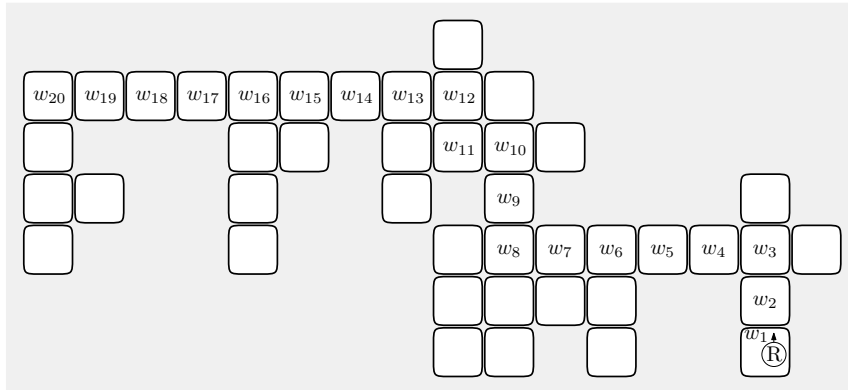
Whenever the orientation is not important or clear from the context we will refer to a left-free or right-free segment by saying *segment*. Note that not every special cell terminates the above sequence of moves.

We now define a *search walk* \mathcal{W} of the robot in an arbitrary component \mathcal{C} (cf. the example in Fig. 2). A search walk depends on the initial position of the robot in \mathcal{C} and on its orientation. We make two assumptions in the definition: the robot is initially located at a full cell of \mathcal{C} and, if $|\mathcal{C}| > 1$, then there is a full cell in front of the robot. The search walk \mathcal{W} is a concatenation of segments. The first segment is both left-free and right-free. If the cell c at the end of this segment is a leaf, then the construction of \mathcal{W} is completed. Otherwise, note that there is a full cell to the left of the robot located at c or to the right of it. In the former case, the search walk is called *left-oriented* and in the latter it is *right-oriented*. Intuitively, a left-oriented search walk prescribes going straight until it is possible to go left, then going straight until it is possible to go right, and so on, alternating directions, until a stop condition is satisfied. A similar intuition concerns right-oriented search walks.

More formally, if $|\mathcal{C}| > 1$, then the search walk \mathcal{W} consists of a single cell. Otherwise, in a right-oriented (respectively left-oriented) search walk, the segments are sequentially added to \mathcal{W} , cyclically alternating the following construction steps.

- (W1) The robot traverses a right-free (respectively left-free) segment, adding it to \mathcal{W} . This segment becomes the last segment in \mathcal{W} if it is a terminal. If the segment is not the last one, then the robot turns right (respectively left).

- (W2) The robot traverses a left-free (respectively right-free) segment, adding it to \mathcal{W} . This segment becomes the last segment in \mathcal{W} if it is a terminal. If the segment is not the last one, then the robot turns left (respectively right).



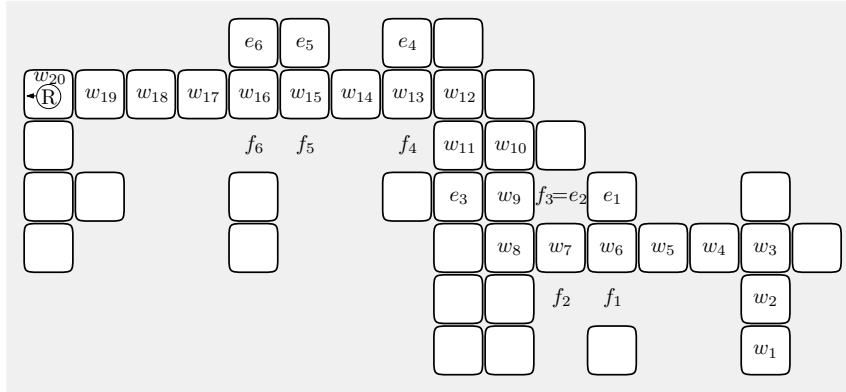
■ **Figure 2** An example of a search walk $\mathcal{W} = (w_1, \dots, w_{20})$ that is constructed by the robot initially located at w_1 and facing North. This search walk is left-oriented, and has three left-free segments $S_1 = (w_1, w_2, w_3)$, $S_3 = (w_8, w_9, w_{10})$, $S_5 = (w_{11}, w_{12})$ and three right-free segments $S_2 = (w_3, \dots, w_8)$, $S_4 = (w_{10}, w_{11})$, $S_6 = (w_{12}, \dots, w_{20})$.

3.2.2 Ensuring the return from a search walk

We start with a high-level idea of the mechanism that will ensure the return from a search walk. Whenever the robot follows a search walk \mathcal{W} , it may a priori not be able to return to the origin of \mathcal{W} . This is due to the fact that, e.g., if \mathcal{W} is left-oriented, then any segment that is right-free may have an unbounded number of special cells such that each of them is adjacent to a cell that does not belong to \mathcal{W} . Thus, the returning robot is not able to remember, using its bounded memory, which of such cells do not belong to the search walk and should be skipped. To overcome this difficulty, the robot will make small changes in the field close to the search walk while traversing it for the first time. These changes may disconnect the component in which the robot is walking, and this may result in creating many new components. While doing so, we will ensure two properties. First, thanks to the modifications in the field performed while traversing \mathcal{W} , the robot is able to return to the first cell of this search walk. Second, while backtracking on \mathcal{W} , the robot is able to undo earlier changes and recover the connectivity of the component.

Each cell c at which the robot stops to perform the above-mentioned modification will be called a *break point* and is defined as follows. First, we require that c be an internal cell of a segment S , i.e., neither the first nor the last cell of S . Second, if S is left-free (respectively right-free), then when the robot traversing S is at c , there is a full cell f to the right (respectively left) of it. Clearly, the cell e to the left (respectively right) of the robot is empty. The following couple of actions performed by the robot located at such a cell c are called a *switch*: if the robot is not carrying a brick, then the robot brings the brick from f and then places it at e , and if the robot is carrying a brick, then the robot places it at e and then brings the brick from f . Note that the switch may disconnect the component in which the robot is located thus creating two new components. One new component is the one in which the robot is located and this is the component that contains the search walk. The second new component is the one that contains the cell f' adjacent to f and at

distance two from c , if f' is full. If the cell f' is full and belongs to a separate component, then this second component containing f' will be called a *switch-component*. Whenever the robot traversing a segment performs the switch at each internal special cell of the segment, we say that this is a *switch-traversal* (cf. the example in Fig. 3).



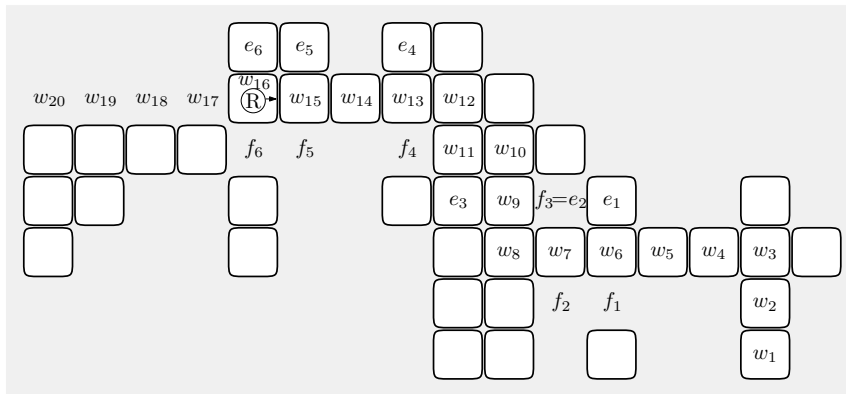
■ **Figure 3** The field from Figure 2 after switch-traversal of the search walk from Figure 2. The cells $w_6, w_7, w_9, w_{13}, w_{15}$ and w_{16} are the break points at which the robot moves a brick from a cell f_i to e_i for $i \in \{1, \dots, 6\}$. Note that a brick is moved from f_2 to e_2 while traversing the second segment and then the same brick is moved to e_3 while traversing the third segment.

3.2.3 Obtaining a brick at the end of a search walk

Informally, the purpose of traversing the entire search walk by a robot is to arrive at a location in the current component \mathcal{C} of the robot, where the robot can start a procedure aimed at obtaining a brick whose removal will not disconnect \mathcal{C} . We will say that such a brick is *free*. In our algorithm, we check the condition (S1) to learn if the last segment is terminal. According to the condition, the terminal segment may end with a leaf, and in such a case the robot is at a cell with a free brick. If the terminal segment does not end with a leaf, then there need not exist a free brick located in a close neighborhood of the robot. However, we will prove that it is possible to perform a series of changes to the field that results in creating a configuration of bricks that does contain a free brick.

We now define the behavior of the robot that ended the switch-traversal of the last segment S of a search walk \mathcal{W} and arrived at a cell that is not a leaf. The following series of moves is called *shifting* (cf. Fig. 4). Suppose that the cell to the right (respectively left) of the robot is full. Note that this implies that S is left-free (respectively right-free). First the robot changes its direction so that a cell of S is in front of it (i.e., the robot turns back). The following three actions are performed until the stop condition specified in the third action occurs. First, the robot picks the brick from the currently occupied cell. Second, the robot moves one step forward – thus backtracking along S . Third, when the robot is at a special cell, then the shifting is completed, and otherwise the robot places the brick at the cell to the left (respectively right) of it. Note that when the robot arrives at a special cell, it is carrying a brick and this is the desired free brick.

We now give the pseudo-code of procedure FindNextBrick that obtains this brick.



■ **Figure 4** The field from Fig. 3 at the end of shifting. The shifting ends with a right-free segment, at the cell w_{16} because it has a full neighbor, the cell e_6 . There is one fewer brick than in Fig. 3 and this is the free brick obtained and carried by the robot.

■ **Procedure FindNextBrick** finding a free brick.

-
- 1 $\mathcal{W} :=$ the search walk that starts at the cell where the robot is located
 - 2 go to the nearest cell belonging to a free component
 - 3 perform a switch-traversal of \mathcal{W}
 - 4 **if** the robot is at a leaf **then**
 - 5 pick the brick
 - 6 **else**
 - 7 perform shifting
-

3.3 Back to the marker

Before presenting the high-level idea of procedure ReturnToMarker that takes the robot carrying a brick back to the marker, we need the following definitions. If S is a segment, then the *reversal* of S , denoted by $\varphi(S)$, is the segment composed of the same cells as S but in the reversed order. For a search walk \mathcal{W} that is a concatenation of segments S_1, \dots, S_l , define the *reversal* of \mathcal{W} , denoted by $\varphi(\mathcal{W})$, to be the walk that is the concatenation of segments $\varphi(S_l), \varphi(S_{l-1}), \dots, \varphi(S_1)$, in this order. Thus, following $\varphi(\mathcal{W})$ means backtracking along \mathcal{W} , and in this section we give a procedure performing it, that reconnects the previous free component on the way. We also define the orientation of $\varphi(\mathcal{W})$ as follows. If the last segment of \mathcal{W} is left-free, then $\varphi(\mathcal{W})$ is left-oriented, and otherwise $\varphi(\mathcal{W})$ is right-oriented. Thus, if \mathcal{W} ended with a left-free (respectively right-free) segment, then $\varphi(\mathcal{W})$ also starts with a left-free (respectively right-free) segment.

At a high level, the robot will perform a switch-traversal along $\varphi(\mathcal{W})$, stopping at each break point to reconnect the corresponding switch-component with the component in which the robot is walking. However, we need to ensure that, at the end, the robot stops at the right point, i.e., at the marker. In order to ensure this, we define the following condition:

(S1') the robot arrives at a cell at distance at most 4 from the marker.

We define a *return switch-traversal* of $\varphi(\mathcal{W})$ to be a switch-traversal of $\varphi(\mathcal{W})$ in which each verification of condition (S1) is replaced by the verification of condition (S1'). Recall that the condition (S1) is checked in the definition of a switch-traversal to determine the termination of a segment and consequently the termination of the entire search walk. Intuitively, by replacing condition (S1) with (S1') we change the behavior of the robot so that it is looking for the marker while backtracking along \mathcal{W} , i.e., going along $\varphi(\mathcal{W})$.

35:12 Building a Nest by an Automaton

A high-level sketch of procedure `ReturnToMarker` is the following. As indicated earlier, the robot essentially follows $\varphi(\mathcal{W})$ and, as the return switch-traversal dictates, reconnects the switch components. However, there is one special case in which the robot should not perform a switch while being at a cell c' of $\varphi(\mathcal{W})$, although c' satisfies the definition of a break point. This case occurs if the shifting moved all internal cells of the last segment of \mathcal{W} . In this case, it is enough for the robot to move to the next cell after c' and start the return switch-traversal of $\varphi(\mathcal{W})$ from there. This is feasible because the cell c and its neighbors are at a bounded distance from the robot when the shifting is completed. Below is the pseudo-code of procedure `ReturnToMarker`.

■ **Procedure `ReturnToMarker`** going back to the marker.

```
1  $\mathcal{W} :=$  the search walk traversed in the last call to FindNextBrick
2 let  $S_1, \dots, S_l$  be the segments in  $\mathcal{W}$ 
3 if the robot is at the first cell of  $S_l$  then
4   turn towards the penultimate cell of  $S_{l-1}$ 
5   move  $\min\{2, |S_{l-1}| - 1\}$  cells forward
6   if  $|S_{l-1}| = 2$  and  $l > 2$ , then make a turn towards the penultimate cell of  $S_{l-2}$ 
7 starting at the current location, perform a return switch-traversal of  $\varphi(\mathcal{W})$ 
8 go to the marker
```

3.4 Extending the rough disc

The aim of procedure `ExtendRoughDisc` is double: it adds a new brick to the current rough disc in a specific place, and it calls procedure `Sweep` to extend, if necessary, the empty space around the rough disc and to ensure that the marker is close to some free component. Whenever procedure `ExtendRoughDisc` is called, the following conditions will be satisfied: the robot is at the marker and it is carrying a brick.

Below is the pseudo-code of procedure `ExtendRoughDisc`.

■ **Procedure `ExtendRoughDisc`** adding one brick to the rough disc D .

```
1 place the brick at the unique cell  $e$  such that  $D \cup \{e\}$  is a rough disc
2 call procedure Sweep
```

Now the pseudo-code of the entire algorithm can be succinctly formulated as follows.

■ **Algorithm `Nest`** building a nest from any connected field.

```
1 if the span of the field is at most 2 then
2   exit ▷ the field is already a nest
3 the cell occupied by the robot becomes the marker
4 a full cell at distance 2 from the marker becomes the initial rough disc
5 call procedure Sweep
6 while there exists a free component  $\mathcal{C}$  do
7   call procedure FindNextBrick
8   call procedure ReturnToMarker
9   call procedure ExtendRoughDisc ▷ moves the marker if necessary
10 pick the brick (the marker) and place it at the unique cell  $e$  of the rough disc  $D$  such
    that  $D \cup \{e\}$  is a rough disc
```

The following is the main result of this paper. Its proof is omitted due to space limitation.

► **Theorem 3.** *Algorithm Nest builds a nest starting from any connected field of size z and span s in time $O(sz)$. This time is worst-case optimal.*

4 Conclusion

We designed a finite deterministic automaton that builds the most compact structure starting from any connected field of bricks, and does it in optimal time. An interesting problem yielded by our research is to characterize the classes of target structures that can be built by a single automaton, starting from any connected field of bricks in the grid. Another problem is that of how the building task parallelizes, i.e., how much time many automata use to build some structure.

References

- 1 Susanne Albers and Monika Rauch Henzinger. Exploring Unknown Environments. *SIAM J. Comput.*, 29(4):1164–1188, 2000. doi:10.1137/S009753979732428X.
- 2 Baruch Awerbuch, Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal Graph Exploration by a Mobile Robot. *Inf. Comput.*, 152(2):155–172, 1999. doi:10.1006/inco.1999.2795.
- 3 Michael A. Bender, Antonio Fernández, Dana Ron, Amit Sahai, and Salil P. Vadhan. The Power of a Pebble: Exploring and Mapping Directed Graphs. *Inf. Comput.*, 176(1):1–21, 2002. doi:10.1006/inco.2001.3081.
- 4 Michael A. Bender and Donna K. Slonim. The Power of Team Exploration: Two Robots Can Learn Unlabeled Directed Graphs. In *35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 75–85, 1994. doi:10.1109/SFCS.1994.365703.
- 5 Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal Learning of an Unknown Environment. *Machine Learning*, 18(2-3):231–254, 1995. doi:10.1007/BF00993411.
- 6 Manuel Blum and Dexter Kozen. On the Power of the Compass (or, Why Mazes Are Easier to Search than Graphs). In *19th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 132–142, 1978. doi:10.1109/SFCS.1978.30.
- 7 Manuel Blum and William J. Sakoda. On the Capability of Finite Automata in 2 and 3 Dimensional Space. In *18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 147–161, 1977. doi:10.1109/SFCS.1977.20.
- 8 Sebastian Brandt, Jara Uitto, and Roger Wattenhofer. A Tight Lower Bound for Semi-Synchronous Collaborative Grid Exploration. In *32nd International Symposium on Distributed Computing (DISC)*, pages 13:1–13:17, 2018. doi:10.4230/LIPIcs.DISC.2018.13.
- 9 Lothar Budach. Automata and labyrinths. *Math. Nachrichten*, 86:195–282, 1978.
- 10 Jérémie Chalopin, Shantanu Das, and Adrian Kosowski. Constructing a Map of an Anonymous Graph: Applications of Universal Sequences. In *14th International Conference on Principles of Distributed Systems (OPODIS)*, pages 119–134, 2010. doi:10.1007/978-3-642-17653-1_10.
- 11 Lihi Cohen, Yuval Emek, Oren Louidor, and Jara Uitto. Exploring an Infinite Space with Finite Memory Scouts. In *28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 207–224, 2017. doi:10.1137/1.9781611974782.14.
- 12 Shantanu Das, Paola Flocchini, Nicola Santoro, and Masafumi Yamashita. On the computational power of oblivious robots: forming a series of geometric patterns. In *29th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 267–276, 2010. doi:10.1145/1835698.1835761.
- 13 Xiaotie Deng and Christos H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, 1999. doi:10.1002/(SICI)1097-0118(199911)32:3<265::AID-JGT6>3.0.CO;2-8.

- 14 Yoann Dieudonné, Franck Petit, and Vincent Villain. Leader Election Problem versus Pattern Formation Problem. In *24th International Symposium on Distributed Computing (DISC)*, pages 267–281, 2010. doi:10.1007/978-3-642-15763-9_26.
- 15 Krzysztof Diks, Pierre Fraigniaud, Evangelos Kranakis, and Andrzej Pelc. Tree exploration with little memory. *J. Algorithms*, 51(1):38–63, 2004. doi:10.1016/j.jalgor.2003.10.002.
- 16 Gregory Dudek, Michael Jenkin, Evangelos E. Milios, and David Wilkes. Robotic exploration as graph construction. *IEEE Trans. Robotics and Automation*, 7(6):859–865, 1991. doi:10.1109/70.105395.
- 17 Christian A. Duncan, Stephen G. Kobourov, and V. S. Anil Kumar. Optimal constrained graph exploration. *ACM Trans. Algorithms*, 2(3):380–402, 2006. doi:10.1145/1159892.1159897.
- 18 Yuval Emek, Tobias Langner, David Stolz, Jara Uitto, and Roger Wattenhofer. How many ants does it take to find the food? *Theor. Comput. Sci.*, 608:255–267, 2015. doi:10.1016/j.tcs.2015.05.054.
- 19 Pierre Fraigniaud and David Ilcinkas. Digraphs Exploration with Little Memory. In *21st Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 246–257, 2004. doi:10.1007/978-3-540-24749-4_22.
- 20 Pierre Fraigniaud, David Ilcinkas, Guy Peer, Andrzej Pelc, and David Peleg. Graph exploration by a finite automaton. *Theor. Comput. Sci.*, 345(2-3):331–344, 2005. doi:10.1016/j.tcs.2005.07.014.
- 21 A. Hemmerling. Labyrinth Problems: Labyrinth-Searching Abilities of Automata. *Teubner-Texte zur Mathematik. B. G. Teubner Verlagsgesellschaft, Leipzig*, 114, 1989.
- 22 Frank Hoffmann. One Pebble Does Not Suffice to Search Plane Labyrinths. In *Fundamentals of Computation Theory (FCT)*, pages 433–444, 1981. doi:10.1007/3-540-10854-8_47.
- 23 Dexter Kozen. Automata and planar graphs. In *Fundamentals of Computation Theory (FCT)*, pages 243–254, 1979.
- 24 Tobias Langner, Barbara Keller, Jara Uitto, and Roger Wattenhofer. Overcoming Obstacles with Ants. In *19th International Conference on Principles of Distributed Systems (OPODIS)*, pages 9:1–9:17, 2015. doi:10.4230/LIPIcs.OPODIS.2015.9.
- 25 Petrisor Panaite and Andrzej Pelc. Exploring Unknown Undirected Graphs. *J. Algorithms*, 33(2):281–295, 1999. doi:10.1006/jagm.1999.1043.
- 26 N. Rao, S. Karetí, W. Shi, and S. Iyengar. Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. Technical Report ORNL/TM-12410, Oak Ridge National Lab., 1993.
- 27 Hans-Anton Rollik. Automaten in planaren Graphen. *Acta Informatica*, 13:287–298, 1980. doi:10.1007/BF00288647.
- 28 Ichiro Suzuki and Masafumi Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *SIAM J. Comput.*, 28(4):1347–1363, 1999. doi:10.1137/S009753979628292X.

Robustness of Randomized Rumour Spreading

Rami Daknama

Department of Mathematics, Ludwig-Maximilians-Universität München, Germany

Konstantinos Panagiotou

Department of Mathematics, Ludwig-Maximilians-Universität München, Germany

Simon Reisser

Department of Mathematics, Ludwig-Maximilians-Universität München, Germany

Abstract

In this work we consider three well-studied broadcast protocols: *push*, *pull* and *push&pull*. A key property of all these models, which is also an important reason for their popularity, is that they are presumed to be very robust, since they are simple, randomized, and, crucially, do not utilize explicitly the global structure of the underlying graph. While sporadic results exist, there has been no systematic theoretical treatment quantifying the robustness of these models. Here we investigate this question with respect to two orthogonal aspects: (adversarial) modifications of the underlying graph and message transmission failures.

We explore in particular the following notion of *local resilience*: beginning with a graph, we investigate up to which fraction of the edges an adversary may delete at each vertex, so that the protocols need significantly more rounds to broadcast the information. Our main findings establish a separation among the three models. On one hand *pull* is robust with respect to all parameters that we consider. On the other hand, *push* may slow down significantly, even if the adversary is allowed to modify the degrees of the vertices by an arbitrarily small positive fraction only. Finally, *push&pull* is robust when no message transmission failures are considered, otherwise it may be slowed down.

On the technical side, we develop two novel methods for the analysis of randomized rumour spreading protocols. First, we exploit the notion of self-bounding functions to facilitate significantly the round-based analysis: we show that for any graph the variance of the growth of informed vertices is bounded by its expectation, so that concentration results follow immediately. Second, in order to control adversarial modifications of the graph we make use of a powerful tool from extremal graph theory, namely Szemerédi's Regularity Lemma.

2012 ACM Subject Classification Mathematics of computing → Probabilistic algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases Rumour Spreading, Local Resilience, Robustness, Self-bounding Functions, Szemerédi's Regularity Lemma

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.36

Related Version A full version of the paper is available at <https://arxiv.org/abs/1902.07618>.

1 Introduction

Randomized broadcast protocols are highly relevant for data distribution in large networks of various kinds, including technological, social and biological networks. Among many others there are three basic models in the literature, introduced in [19, 9, 24], namely *push*, *pull* and *push&pull* (or short *pp*). Consider a connected graph in which some vertex holds a piece of information; we call this vertex (initially) informed. All three models have the common characteristic that they proceed in rounds. In the *push* model, in every round every informed vertex chooses a neighbour independently and uniformly at random (iuar) and informs it; this of course has only an effect if the target vertex was previously uninformed. Contrary, in the *pull* model every round every uninformed vertex chooses a neighbour iuar and asks for the information. If the asked vertex has the information, then the asking vertex becomes



© Rami Daknama, Konstantinos Panagiotou, and Simon Reisser;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 36; pp. 36:1–36:15



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

informed as well. The third model *push&pull* combines both worlds: in each round, each vertex chooses a neighbour *uar*, and if one of both vertices is informed, then afterwards both become so. We additionally assume that each message transmission succeeds independently with probability $q \in (0, 1]$. For these algorithms, the main parameter that we consider is the random variable that counts how many rounds are needed until all vertices are informed, and we call these quantities the *runtimes* of the respective algorithms.

In the remainder we will denote the runtime of *push* by $T_{push}(G, v, q)$ where G is the underlying graph, initially the vertex v is informed and we have a transmission success probability of $q \in (0, 1]$. Analogously we denote the runtimes of *pull* and *push&pull* by $T_{pull}(G, v, q)$ and $T_{pp}(G, v, q)$ respectively. If the choice of v does not matter we will omit it in our notation. The most basic case is when G is the complete graph K_n with n vertices. Then, see for example Doerr and Kostrygin [11], it is known that for $\mathcal{P} \in \{push, pull, pp\}$ and $q \in (0, 1]$ in expectation and with probability tending to 1 as $n \rightarrow \infty$

$$T_{\mathcal{P}}(K_n, q) = c_{\mathcal{P}}(q) \log n + o(\log n),$$

where, for $q \in (0, 1)$,

$$c_{push}(q) := \frac{1}{\log(1+q)} + \frac{1}{q}, \quad c_{pull}(q) := \frac{1}{\log(1+q)} - \frac{1}{\log(1-q)},$$

$$c_{pp}(q) := \frac{1}{\log(1+2q)} + \frac{1}{q - \log(1-q)},$$

and where we set $c_{\mathcal{P}}(1) := \lim_{q \rightarrow 1} c_{\mathcal{P}}(q)$. If q is clear from the context, we write $c_{\mathcal{P}}$ instead of $c_{\mathcal{P}}(q)$. Actually, the results in [11] and also [12] are much more precise, but the stated forms will be sufficient for what follows.

Contribution & Related Work

In this article our focus is on quantifying the *robustness* of all three models. Indeed, robustness is a key property that is often attributed to them, since they are simple, randomized, and, crucially, do not exploit explicitly the structure of the underlying graph (apart, of course, from considering the neighborhoods of the vertices). Clearly, the runtime can vary tremendously between different graphs with the same number of vertices. Hence it is essential to understand which structural characteristics of a graph influence in what way the runtime of rumour spreading algorithms.

One result in this spirit for the *push* model was shown in [25]. Roughly speaking, in that paper it is shown that even on graphs with low density, if the edges are distributed rather uniformly, then *push* is as fast as on the complete graph. This can be interpreted as a robustness result: starting with a complete graph, one can delete a vast amount of edges and as long as this is done rather uniformly, the runtime of *push* is affected insignificantly. To state the result more precisely, we need the following notion.

► **Definition 1** ($(n, \delta, \Delta, \lambda)$ -graph). *Let G be a connected graph with n vertices that has minimum degree δ and maximum degree Δ . Let $\mu_1 \geq \mu_2 \geq \dots \geq \mu_n$ be the eigenvalues of the adjacency matrix of G , and set $\lambda = \max_{2 \leq i \leq n} |\mu_i| = \max\{|\mu_2|, |\mu_n|\}$. We will call G an $(n, \delta, \Delta, \lambda)$ -graph.*

In this paper we are interested in the case where G gets large, that is, when $n \rightarrow \infty$. Hence all asymptotic notation in this paper is with respect to n ; in particular “with high probability”, or short whp, means with probability $1 - o(1)$ when $n \rightarrow \infty$.

► **Definition 2** (Expander Sequence). Let $\mathcal{G} = (G_n)_{n \in \mathbb{N}}$ be a sequence of graphs, where G_n is a $(n, \delta_n, \Delta_n, \lambda_n)$ -graph for each $n \in \mathbb{N}$. We say that \mathcal{G} is an expander sequence if $\Delta_n/\delta_n = 1 + o(1)$ and $\lambda_n = o(\Delta_n)$.

Note that if we consider any sequence $\mathcal{G} = (G_n)_{n \in \mathbb{N}}$ of graphs this always implicitly defines δ_n, Δ_n and λ_n as in Definition 2. Expander graphs have found numerous applications in computer science and mathematics, see for example the survey [23]. If \mathcal{G} is an expander sequence, then intuitively this means that for n large enough, the edges of G_n are rather uniformly distributed. For a more formal statement see Lemma 16. Moreover, note that our definition of expander sequences excludes the case when Δ_n is bounded; this is actually a necessary condition for our robustness results to hold, see [13]. With all these definitions at hand we can state the result from [25] that quantifies the robustness of *push* with respect to the network topology, that is, the runtime is asymptotically the same as on the complete graph K_n .

► **Theorem 3.** Let $\mathcal{G} = (G_n)_{n \in \mathbb{N}}$ be an expander sequence. Then whp

$$T_{push}(G_n) = c_{push}(1) \log n + o(\log n).$$

Apart from expander sequences, results in the form of Theorem 3 (where the asymptotic runtimes of one or more of these algorithms are determined) were also shown for sufficiently dense Erdős-Renyi random graphs [16], random regular graphs [15] as well as hypercubes [25]. Moreover, the order of the runtime on various models that describe social networks was investigated. In [17] the Chung-Lu model was studied, [10] explored preferential attachment graphs and [18] examined geometric graphs. A somewhat different approach is to derive general runtime bounds that hold for all graphs and depend only on some graph parameter, e.g. conductance [20, 6], vertex expansion [21] or diameter [14, 5, 22]. Furthermore, several variants of *push*, *pull* and *push&pull* were studied. These include vertices being restricted to answer only one *pull* request per round [7], vertices being allowed to contact multiple neighbours per round [25, 11], vertices not calling the same neighbour twice [10] and asynchronous versions [4, 26, 1, 2]. Finally, besides [11], robustness of these rumor spreading algorithms with respect to message transmission failures was also studied by Elsässer and Sauerwald in [13]. It was shown for any graph that if a message fails with probability $1 - p$, then the runtime of *push* increases at most by a factor of $6/p$.

In this work our focus is on three subjects concerning the robustness of rumour spreading. Our first (and not unexpected) result extends Theorem 3 to the runtimes of *pull* and *push&pull*. In particular, we show that none of the three protocols slows down or speeds up on graphs with good expansion properties compared to its runtime on the complete graph. This motivates to investigate how severely a graph with good expansion properties has to be modified to increase the respective runtimes.

In our second contribution, which is also the main result and which differs from what was treated in previous works, we propose and study a novel approach to quantifying robustness. In particular, we investigate the impact of adversarial edge deletions, where we use the well-known concept of *local resilience*, see e.g. [28, 8]. To be specific, we explore up to which fraction of edges an adversary may delete at each vertex to slow down the process by a significant amount of time, i.e., by $\Omega(\log n)$ rounds. Here we discover a surprising dichotomy in the following sense. On the one hand, we show that both *pull* and *push&pull* cannot be slowed down by such adversarial edge deletions – in essentially all but trivial cases, where the fraction is so large that the graph may become (almost) disconnected. On the other hand, we demonstrate that even a small number of edge deletions is sufficient to slow down

push by $\Omega(\log n)$ rounds. In other words, we find that in contrast to *pull* and *push&pull*, the *push* protocol is not resilient to adversarial deletions and lacks (in this specific sense) the robustness of the other two protocols.

As our third subject, we generalise the previous results by additionally considering message transmission failures that occur independently with probability $1 - q \in [0, 1)$. On the positive side, we show that for arbitrary $q \in (0, 1]$ all three algorithms inform *almost* all vertices at least as fast as when run on expander sequence in spite of adversarial edge deletions. However, if we want to inform all vertices, only *pull* is not slowed down by adversarial edge deletions for all values of q ; *push* can be slowed down as before; and *push&pull* is a mixed bag, for $q = 1$ it cannot be slowed down, for $q < 1$ it can. Furthermore, in general it is also possible to speed *push&pull* up by deleting edges, which is however not surprising as the star-graph deterministically finishes in at most 2 rounds.

Summarizing, this work expands previous (robustness) results, particularly the ones concerning precise asymptotic runtimes and random transmission failures. Crucially, we introduce and study the concept of local resilience as a method to investigate robustness. However, apart from that, in this paper we develop two new general methods for the analysis of rumour spreading algorithms.

- The most common approach in the current literature for the study of the runtime is to determine the expected number of newly informed vertices in one or more rounds and to show concentration, for example by bounding the variance. Achieving this, however, is often quite complex and makes laborious and lengthy technical arguments necessary. Here we use the theory of *self-bounding* functions, see Section 2, that allows us to cleanly upper bound the variance by the *expected value*. The argument works for all three investigated algorithms and the bound is valid for all graphs. We are certain that this method will also facilitate future work on the analysis of rumour spreading algorithms.
- Studying the robustness of the protocols is a challenging task, as the adversary (as described previously) has various options to modify the graph, for example by introducing a high variance in the degrees of the vertices; this turns out to be particularly problematic in the case of *push&pull*. Here we demonstrate that such types of irregularities can be handled universally by applying a powerful tool from a completely different area, namely extremal graph theory. In particular, we use Szemerédi’s regularity lemma (see e.g. [27]), which allows us to partition the vertex set of a graph such that nearly all pairs of sets in the partition behave nearly like perfect regular bipartite graphs. This allows us to apply our methods on these regular pairs; eventually we obtain a linear recursion that can be solved by analysing the maximal eigenvalue of the underlying matrix.

1.1 Results

Our first result addresses the question about how fast rumours spread on expander graphs; in order to obtain a concise statement also the occurrence of independent message transmission failures is considered.

► **Theorem 4.** *Let $\mathcal{G} = (G_n)_{n \in \mathbb{N}}$ be an expander sequence and let $q \in (0, 1]$. Then whp*

- (a) $T_{push}(G_n, q) = c_{push}(q) \log n + o(\log(n))$,
- (b) $T_{pull}(G_n, q) = c_{pull}(q) \log n + o(\log(n))$,
- (c) $T_{pp}(G_n, q) = c_{pp}(q) \log n + o(\log(n))$.

The first statement is an extension of Theorem 3 and its proof is a straightforward adaptation of the proof in [25]. We omit it. The contribution here is the proof of (b) and (c). Next we consider the case with edge deletions in addition to the message transmission failures.

► **Theorem 5.** *Let $0 < \varepsilon < 1/2, q \in (0, 1]$ and $\mathcal{G} = (G_n)_{n \in \mathbb{N}}$ be an expander sequence. Let $\tilde{\mathcal{G}} = (\tilde{G}_n)_{n \in \mathbb{N}}$ be such that each \tilde{G}_n is obtained by deleting edges of G_n such that each vertex keeps at least a $(1/2 + \varepsilon)$ fraction of its edges. Then whp*

(a) $T_{pull}(\tilde{G}_n, q) = c_{pull}(q) \log n + o(\log n)$.

(b) $T_{pp}(\tilde{G}_n, 1) \leq c_{pp}(1) \log n + o(\log n)$, when additionally assuming that $\delta(G_n) \geq \alpha n$ for some constant $0 < \alpha \leq 1$.

This result demonstrates unconditionally the robustness of *pull*, and conditionally on $q = 1$ the robustness of *push&pull* on dense graphs, in the case of edge deletions, that is, the runtime is asymptotically the same as in the complete graph. It even shows that *push&pull* may potentially profit from edge deletions in contrast to being slowed down. The proof of this result, especially the statement about *push&pull*, is rather involved, since the original graph may become quite irregular after the edge deletions. Here we use, among many other ingredients, the aforementioned decomposition of the graph given by Szemerédi's regularity lemma.

Note that Theorem 5 does not consider *push* and *push&pull* (when $q \neq 1$) at all. Indeed, our next result states that in these cases the behaviour is rather different and that the algorithms may be slowed down.

► **Theorem 6.** *Let $\varepsilon > 0$ and $q \in (0, 1]$. Then there is an expander sequence $\mathcal{G} = (G_n)_{n \in \mathbb{N}}$ and a sequence of graphs $\tilde{\mathcal{G}} = (\tilde{G}_n)_{n \in \mathbb{N}}$ with the following properties. Each \tilde{G}_n is obtained by deleting edges of G_n such that each vertex keeps at least a $(1 - \varepsilon)$ fraction of its edges. Moreover, whp*

(a) $T_{push}(\tilde{G}_n, q) \geq c_{push}(q) \log n + \varepsilon/(2q) \log n + o(\log n)$.

(b) $T_{pp}(\tilde{G}_n, q) \geq c_{pp}(q) \log n + (\varepsilon/(8q) - \varepsilon q^3/5) \log n + o(\log n)$.

Nevertheless, not all hope is lost. On the positive side, the next result states that *push* and *push&pull* are able to inform *almost* all vertices as fast as on the complete graph in spite of adversarial edge deletions. In this sense, we obtain an almost-robustness result for these cases.

► **Theorem 7.** *Let $0 < \varepsilon < 1/2, q \in (0, 1]$ and $\mathcal{G} = (G_n)_{n \in \mathbb{N}}$ be an expander sequence. Let $\tilde{\mathcal{G}} = (\tilde{G}_n)_{n \in \mathbb{N}}$ be such that each \tilde{G}_n is obtained by deleting edges of G_n such that each vertex keeps at least a $(1/2 + \varepsilon)$ fraction of its edges. For $\mathcal{P} \in \{\text{push}, \text{pp}\}$ let $\tilde{T}_{\mathcal{P}}$ denote the number of rounds needed to inform at least $n - n/\log n$ vertices. Then whp*

(a) $\tilde{T}_{push}(\tilde{G}_n) = \log_{1+q}(n) + o(\log n)$.

(b) $\tilde{T}_{pp}(\tilde{G}_n) \leq \log_{1+2q}(n) + o(\log n)$, when additionally assuming that $\delta(G_n) \geq \alpha n$ for some constant $0 < \alpha \leq 1$.

We conjecture that there is also a version of Theorem 7b that is true for *push&pull* on sparse graphs; to be precise we conjecture that in the setting of Theorem 7b it is $\tilde{T}_{pp}(\tilde{G}_n) \leq \log_{1+2q}(n) + o(\log n)$, without further restrictions on G_n , i.e. that *push&pull* cannot be slowed down informing *almost* all vertices.

As a final remark note that Theorems 5 and 7 are tight in the sense that if an adversary may delete up to half of the edges at each vertex, then there are expander graphs that become disconnected. On those graphs a linear fraction of the vertices will remain uninformed forever.

Outline

The rest of this paper is structured as follows. The first part of Section 2 contains our technical contribution concerning the analysis through self-bounding functions. In the second part we state the Expander Mixing Lemma and give some applications to our setting with

deleted edges. The remaining sections contain the proofs to the main theorems. The proof of Theorem 4 has two steps: determining the expected growth rates of the number of informed vertices after performing one round, then concluding the proof for the runtime by using the tools developed in Section 2. This proof is not included in this version, here instead we focus on the case with edge deletions, where for every protocol we use a different method to show the claimed results. In Subsection 3.1 we show that edge deletions do not slow down *pull*, by analysing the number of edges between informed and uninformed vertices. Showing that adversarial edge deletions cannot slow down the time until *push* has informed almost all vertices will be archived in Section 3.2 by giving a coupling to the case without edge deletions. Then, in Subsection 3.3 we show that *push&pull* informs almost all vertices of dense graphs fast in spite of adversarial edge deletions. We utilize a version of Szemerédi Regularity Lemma to get a well-behaved partition of the vertex set that is suitable for performing a round based analysis. However, if $q < 1$, adversarial edge deletions can slow down the time until *push&pull* has informed all vertices for nearly all values of q ; we show this in Section 3.4. The same example as given there also yields Theorem a. Finally, an unabridged version of this paper, that contains any proofs that are omitted here, is available at <https://arxiv.org/abs/1902.07618>.

Further Notation

Let $G = (V, E)$ denote a graph with vertex set V and edge set $E \subseteq \binom{V}{2}$. Consider $v \in V$ and $U, W \subseteq V$ with $U \cap W = \emptyset$. We will denote the set of neighbours of v in G by $N_G(v)$ or by $N(v)$ and we will denote its degree by $d_G(v) := |N_G(v)|$ or by $d(v)$; δ_G or δ and Δ_G or Δ denote minimum and maximum degree of G . Similarly the neighbourhood of any set of vertices $S \subseteq V$ is defined by $N_G(S) := \cup_{v \in S} N_G(v)$. Furthermore let $E(U, W) = E_G(U, W)$ denote the set of edges with one vertex in U and one vertex in W and let $e(U, W) := e_G(U, W) := |E_G(U, W)|$. With $E_G(U)$ we denote the set of edges with both vertices in U ; $e_G(U) = |E_G(U)|$. For any round $t \in \mathbb{N}$ and $\mathcal{P} \in \{\text{push}, \text{pull}, \text{pp}\}$, we denote by $I_t^{(\mathcal{P})}(G)$ the set of vertices of G informed by *push*, *pull* and *push&pull* respectively at the beginning of round t and $|I_1^{(\mathcal{P})}| = 1$; if the underlying graph is clear from the context we will omit it; if we consider a sequence of graphs $\mathcal{G} = (G_n)_{n \in \mathbb{N}}$ and a sequence of times $t = (t(n))_{n \in \mathbb{N}}$, then $I_t^{(\mathcal{P})}(\mathcal{G}) = (I_{t(n)}^{(\mathcal{P})}(G_n))_{n \in \mathbb{N}}$ is also a sequence. Similarly, $U_t^{(\mathcal{P})} := V \setminus I_t^{(\mathcal{P})}$ denotes the set of uninformed vertices. With \log we refer to the natural logarithm. For any event A we will write $\mathbb{E}_t[A]$ instead of $\mathbb{E}[A|I_t]$ for the conditional expectation and $P_t[A]$ instead of $P[A|I_t]$ for the conditional probability. Finally we want to clarify our use of Landau symbols. Let $a, b \in \mathbb{R}$ and f be a function. The terms $a \leq b + o(f)$ and $a \geq b - o(f)$ mean that there exist positive functions $g, h \in o(f)$ such that $a \leq b + g$ and $a \geq b - h$. Consequently $a = b + o(f)$ means that there exists a positive function $g \in o(f)$ such that $a \in [b - g, b + g]$

2 Tools & Techniques

In this section we collect and prove statements about our protocols and properties of expander sequences. We begin with applying the previously mentioned notion of self-bounding functions to derive universal and simple-to-apply concentration results for our random variables, i.e., the number of informed vertices after a particular round. Then we extend the concentration results to more than one round. In the last part we recall the well known Expander Mixing Lemma and utilize it to derive properties (weak expansion, path enumeration) for the case where we delete edges from our graphs.

Self-bounding functions

Our main technical new result in this section is the following bound on the variance for the number of informed vertices in any given round; it is true for any graph and any set of informed vertices.

► **Lemma 8.** *Let G be a graph, $t \in \mathbb{N}$ and $I_t = I_t^{(\mathcal{P})}(G)$ for $\mathcal{P} \in \{\text{push}, \text{pull}, \text{pp}\}$. Then*

$$\text{Var}[|I_{t+1}| | I_t] \leq \mathbb{E}[|I_{t+1}| | I_t].$$

Lemma 8 follows directly from Lemmas 10 and 11. Before stating them we introduce the notion of self-bounding functions.

► **Definition 9** (Self-bounding function). *Let X be a set and $m \in \mathbb{N}$. A non-negative function $f : X^m \rightarrow \mathbb{R}$ is self-bounding, if there exist functions $f_i : X^{m-1} \rightarrow \mathbb{R}$ such that for all $x_1, \dots, x_m \in X$ and all $i = 1, \dots, m$*

$$0 \leq f(x_1, \dots, x_m) - f_i(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_m) \leq 1$$

and

$$\sum_{1 \leq i \leq m} (f(x_1, \dots, x_m) - f_i(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_m)) \leq f(x_1, \dots, x_m).$$

A striking property of self-bounding function is the following bound on the variance.

► **Lemma 10** ([3]). *For a self-bounding function f and independent random variables X_1, \dots, X_m , $m \in \mathbb{N}$*

$$\text{Var}[f(X_1, \dots, X_m)] \leq \mathbb{E}[f(X_1, \dots, X_m)].$$

► **Lemma 11.** *Let G be a graph, $t \in \mathbb{N}$, and let $I_t = I_t^{(\mathcal{P})}(G)$ for $\mathcal{P} \in \{\text{push}, \text{pull}, \text{pp}\}$. Then, conditional on I_t , there exist $m \in \mathbb{N}$, independent random variables X_1, \dots, X_m and a self-bounding function $f = f^{(\mathcal{P})}$ such that $|I_{t+1}| = f(X_1, \dots, X_m)$.*

► **Remark 12.** Let $G = (V, E)$ be a graph. Lemma 11 also applies to subsets of I_{t+1} , i.e. for any $U \subset V$ and conditioned on I_t we have that $|I_{t+1} \cap U|$ and $|(I_{t+1} \cap U) \setminus I_t|$ are self-bounding.

The following lemma gives a tool that we will use in order to extend our round-wise analysis to longer phases.

► **Proposition 13.** *Let $\mathcal{P} \in \{\text{push}, \text{pull}, \text{pp}\}$, $I_t = I_t^{(\mathcal{P})}$ and $t_1 \geq t_0 \geq 1$ such that $|I_{t_0}| \geq \sqrt{\log n}$. Let further $(\mathcal{A}_i)_{i \in \mathbb{N}}$ be a sequence of events, $c > 1$, and $\delta > 0$ such that*

$$P_{t_0}[\mathcal{A}_t | \mathcal{A}_{t_0}, \dots, \mathcal{A}_{t-1}] \geq 1 - \delta (c^{t-t_0} |I_{t_0}|)^{-1/3} \quad \text{for all } t_0 \leq t \leq t_1.$$

Then

$$P_{t_0} \left[\bigcap_{t=t_0}^{t_1} \mathcal{A}_t \right] \geq 1 - O(|I_{t_0}|^{-1/3})$$

We give two typical example applications of this lemma below. The first example addresses the case where we have a lower bound for the expected number of informed vertices after one round.

► **Example 14.** Let $\mathcal{P} \in \{\text{push}, \text{pull}, \text{pp}\}$, $I_t = I_t^{(\mathcal{P})}$. Assume that there is some $c > 1$ such that $\mathbb{E}_t [|I_{t+1}|] \geq c|I_t|$ for all t as long as $n/f(n) \leq |I_t| \leq n/g(n)$ for some functions $1 \leq f, g \leq n$, $f = o(n)$. Let t_0 be such that $|I_{t_0}| \geq n/f(n)$. Then according to Lemma 8 we have that $\text{Var}_t [|I_{t+1}|] \leq \mathbb{E}_t [|I_{t+1}|]$ and applying Chebychev's inequality gives

$$P_t \left[\left| |I_{t+1}| - \mathbb{E}_t [|I_{t+1}|] \right| \leq \mathbb{E}_t [|I_{t+1}|]^{2/3} \right] \geq 1 - \mathbb{E}_t [|I_{t+1}|]^{-1/3} \geq 1 - |I_t|^{-1/3}. \quad (1)$$

Consider the events

$$\mathcal{A}_t = \left[|I_t| \geq \mathbb{E}_{t-1} [|I_t|] - \mathbb{E}_{t-1} [|I_t|]^{2/3} \quad \text{or} \quad |I_t| \geq n/g(n) \right]$$

The intersection of $\mathcal{A}_{t_0+1}, \dots, \mathcal{A}_t$ implies inductively that either $|I_t| \geq n/g(n)$ or

$$|I_t| \geq \left(1 - \mathbb{E}_{t-1} [|I_t|]^{-1/3} \right) \mathbb{E}_{t-1} [|I_t|] \geq \left((1 - (c|I_{t_0}|)^{-1/3})c \right)^{t-t_0} |I_{t_0}|.$$

We obtain with (1)

$$P_{t_0} [\mathcal{A}_{t+1} \mid \mathcal{A}_{t_0+1}, \dots, \mathcal{A}_t, |I_t| < n/g(n)] \geq 1 - \left((1 - (c|I_{t_0}|)^{-1/3})c \right)^{-(t-t_0)/3} |I_{t_0}|^{-1/3},$$

and otherwise $P_{t_0} [\mathcal{A}_{t+1} \mid \mathcal{A}_{t_0+1}, \dots, \mathcal{A}_t, |I_t| \geq n/g(n)] = 1$. Choose $\tau := t - t_0 = \log_c(f(n)/g(n)) + o(\log n)$ as small as possible such that this lower bound for $|I_{t+1}|$ is $\geq n/g(n)$, that is, this lower bound is $< n/g(n)$ for $t = t_0 + \tau$. Combining the two conditional probabilities we obtain for all $t_0 \leq t \leq t_0 + \tau$

$$P_{t_0} [\mathcal{A}_{t+1} \mid \mathcal{A}_{t_0+1}, \dots, \mathcal{A}_t] \geq 1 - \left((1 - (c|I_{t_0}|)^{-1/3})c \right)^{-(t-t_0)/3} |I_{t_0}|^{-1/3}.$$

Applying Proposition 13 then yields whp

$$|I_{t_0+\tau+1}| \geq n/g(n).$$

In the second example we make the stronger assumption that we can determine asymptotically the expected number of informed vertices after one round. Here we assume that we begin with a “small” set of informed vertices, say of size $\sqrt{\log n}$, and want to reach a set of size nearly linear in n .

► **Example 15.** Assume that there is some $c > 1$ such that $\mathbb{E}_t [|I_{t+1}|] = (1 + o(1))c|I_t|$ for all t as long as $\sqrt{\log n} \leq |I_t| \leq n/\log n$. Let \mathcal{A}_t be the event “ $\left| |I_t| - \mathbb{E}_{t-1} [|I_t|] \right| \leq \mathbb{E}_{t-1} [|I_t|]^{2/3}$ ” and let t_0 be such that $|I_{t_0}| \geq \sqrt{\log n}$. There is $h(n) \in o(1)$ such that for $c^- := (1 - h(n))c$ and $c^+ := (1 + h(n))c$ we have that $\mathbb{E}_t [|I_{t+1}|] \leq c^+ |I_t|$ and $\mathbb{E}_t [|I_{t+1}|] \geq c^- |I_t|$. Using this notation, the events $\mathcal{A}_{t_0+1}, \dots, \mathcal{A}_{t+1}$ imply together inductively that

$$|I_{t+1}| \leq \left(1 + \mathbb{E}_t [|I_{t+1}|]^{-1/3} \right) \mathbb{E}_t [|I_{t+1}|] \leq \left((1 + (c^- |I_{t_0}|)^{-1/3})c^+ \right)^{t-t_0} |I_{t_0}|$$

for all t such that the right-hand side is bounded by $n/\log n$. Moreover, for all such t

$$|I_{t+1}| \geq \left(1 - \mathbb{E}_t [|I_{t+1}|]^{-1/3} \right) \mathbb{E}_t [|I_{t+1}|] \geq \left((1 - (c^- |I_{t_0}|)^{-1/3})c^- \right)^{t-t_0} |I_{t_0}|.$$

Thus, as \mathcal{A}_t only depends on I_t it follows with (1)

$$P_{t_0} [\mathcal{A}_{t+1} \mid \mathcal{A}_{t_0+1}, \dots, \mathcal{A}_t] \geq 1 - \left((1 - (c^- |I_{t_0}|)^{-1/3})c^- \right)^{-(t-t_0)/3} |I_{t_0}|^{-1/3}.$$

Applying Proposition 13 then immediately gives that there is $\tau_1 = \log_c(n/|I_{t_0}|) + o(\log n)$ such that whp $|I_{t_0+\tau_1}| \leq n/\log n$. Example 14, setting $f = n/\sqrt{\log n}$ and $g = \log n$, gives an additional $\tau_2 = \log_c(n/|I_{t_0}|) + o(\log n)$ such that $|\tau_1 - \tau_2| = o(\log n)$ and whp

$$|I_{t_0+\tau_1}| \leq \frac{n}{\log n} \leq |I_{t_0+\tau_2}|.$$

Expander Sequences

In this section we collect some important properties of expander sequences that we are going to use later. We start by stating a version of the well-known expander mixing lemma applied to our setting of expander sequences.

► **Lemma 16** ([25, Cor. 2.4]). *Let $\mathcal{G} = (G_n)_{n \in \mathbb{N}} = ((V_n, E_n))_{n \in \mathbb{N}}$ be an expander sequence. Then for $S_n \subseteq V_n$ such that $1 \leq |S_n| \leq n/2$ it is*

$$\left| e(S_n, V_n \setminus S_n) - \frac{\Delta_n |S_n| (n - |S_n|)}{n} \right| = o(\Delta_n) |S_n|.$$

The following result is a consequence of the Expander Mixing Lemma that applies to graphs in which some edges were removed. It seems very simple but it turns out to be surprisingly useful.

► **Lemma 17.** *Let $\mathcal{G} = (G_n)_{n \in \mathbb{N}} = ((V_n, E_n))_{n \in \mathbb{N}}$ be an expander sequence. Let $\varepsilon > 0$ and set $\tilde{\mathcal{G}} = (\tilde{G}_n)_{n \in \mathbb{N}}$, where each \tilde{G}_n it is obtained from G_n by deleting edges such that each vertex keeps at least a $(1/2 + \varepsilon)$ fraction of its edges. For each $n \in \mathbb{N}$ let further $S_n \subseteq V_n$, then there is $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$*

$$e_{\tilde{G}_n}(S_n, V_n \setminus S_n) \geq \varepsilon e_{G_n}(S_n, V_n \setminus S_n).$$

3 Proofs

3.1 Proof of Theorems 4b, 5a – edge deletions do not slow down pull

Let $0 < \varepsilon \leq 1/2$. In this section we study the runtime of *pull* in the case in which the input graph is an expander, and where at each vertex at most an $(1/2 - \varepsilon)$ fraction of the edges is deleted. The runtime on expander sequences without edge deletions, that is, the setting in Theorem 4b, is included as the special case where we set $\varepsilon = 1/2$. In contrast to previous proofs, in the analysis of *pull* the “standard” approach that consists of showing, for example, that $\mathbb{E}_t[|I_{t+1} \setminus I_t|] \approx |I_t|$ fails. The main reason is that the graph between I_t and U_t might be quite irregular, so that, depending on the actual state, $\mathbb{E}_t[|I_{t+1} \setminus I_t|] \approx c|I_t|$ for some $c < 1$. However, we discover a different invariant that is preserved, namely that the number of edges between I_t and U_t behaves in an exponential way. With Lemmas 16 and 17 we can then relate this to the number of informed vertices.

► **Lemma 18.** *Consider the setting of Theorem 5a and let $I_t = I_t^{(pull)}$.*

(a) *Let $\sqrt{\log n} \leq |I_t| \leq n/\log n$. Then $|e(U_{t+1}, I_{t+1}) - (1+q)e(U_t, I_t)| \leq |I_t|^{-1/3} e(U_t, I_t)$ with probability at least $1 - O(|I_t|^{-1/3})$.*

(b) *Let $|U_t| \leq n/\log n$. Then $\mathbb{E}_t[|U_{t+1}|] = (1 - q + o(1))|U_t|$.*

Lemma 19 gives a lower bound, that together with an upper bound provided by Lemma 20 imply Theorems 4b and 5a.

► **Lemma 19** (Upper bound in Theorem 5a). *Consider the setting of Theorem 5a and let $I_t = I_t^{(pull)}$, then the following statements hold whp.*

(a) *Let $\sqrt{\log n} \leq |I_t| \leq n/\log n$. Then there are $\tau_1, \tau_2 = \log_{1+q}(n/|I_t|) + o(\log n)$ such that $|I_{t+\tau_2}| < n/\log n < |I_{t+\tau_1}|$.*

(b) *Let $n/\log n \leq |I_t| \leq n - n/\log n$. Then there is $\tau = o(\log n)$ such that $|I_{t+\tau}| > n - n/\log n$.*

(c) *Let $|I_t| \geq n - n/\log n$.*

1. *Case $q = 1$: Then there is $\tau = o(\log n)$ such that $|I_{t+\tau}| = n$.*

2. *Case $q \neq 1$: Then there is $\tau \leq -\log n / \log(1 - q) + o(\log n)$ such that $|I_{t+\tau}| = n$.*

36:10 Robustness of Randomized Rumour Spreading

Note that for $q = 1$ this already implies Theorems 4b and 5a. This leaves the case for $q \neq 1$.

► **Lemma 20.** *Let $0 < \varepsilon \leq 1/2, q \in (0, 1]$ and $\mathcal{G} = (G_n)_{n \in \mathbb{N}}$ be an expander sequence. Let $\tilde{\mathcal{G}} = (\tilde{G}_n)_{n \in \mathbb{N}}$ be such that each \tilde{G}_n is obtained by deleting edges of G_n such that each vertex keeps at least a $(1/2 + \varepsilon)$ fraction of its edges and abbreviate $I_t = I_t^{(pull)}$. Let further $q \in (0, 1)$ and $|I_t| \leq n/2$. Then for $\tau = -\log n / \log(1 - q)$ and all $c < 1$ whp $|I_{t+c\tau}| < n$.*

3.2 Proof of Theorem 7a – push informs almost all vertices fast in spite of edge deletions

To shorten the notation let us call the setting with deleted edges “new model” and the setting without “old model”, that is, the term new model corresponds to the graphs in $\tilde{\mathcal{G}}$, while old model refers to the (original) graphs in \mathcal{G} . We prove Lemma 21 that directly implies Theorem a. We write $I_t = I_t^{(push)}$ throughout.

► **Lemma 21.** *Under the assumptions of Theorem 7a the following holds for the new model:*

(a) *There are $\tau, \tilde{\tau} = \log_{1+q}(n) + o(\log n)$ such that whp $|I_{\tilde{\tau}}| < n / \log n < |I_{\tau}|$.*

(b) *Assume $|I_t| \geq n / \log n$. Then there is a $\tau = o(\log n)$ such that whp $|I_{t+\tau}| \geq n - n / \log n$.*

For the proof of Lemma 21 we will need the following statements, the first one taken from [25].

► **Lemma 22** (Proof of Lemma 2.5 in [25]). *Consider the old model. Assume $|I_t| < n / \log n$ and $q = 1$. Then*

$$P_t[|I_{t+1}| = |I_t| + (1 - o(1))|I_t|] = 1 - o(1). \quad (2)$$

► **Lemma 23.** *Consider push on a sequence of graphs $(G_n)_{n \in \mathbb{N}}$, where G_n has n vertices. Assume that $|I_t| = \omega(1)$ and that (2) holds for $q = 1$, that is, assume that $P_t[|I_{t+1}| = |I_t| + (1 - o(1))|I_t|] = 1 - o(1)$ for $q = 1$. Then for $q \in (0, 1]$*

$$P_t[|I_{t+1}| = |I_t| + (q - o(1))|I_t|] = 1 - o(1). \quad (3)$$

Moreover, assume that whenever $|I_t| < n / \log n$, for $q = 1$, (2) holds. Then there are $\tau, \tilde{\tau} = \log_{1+q}(n) + o(\log n)$ such that whp

$$|I_{\tilde{\tau}}| < n / \log n < |I_{\tau}|. \quad (4)$$

3.3 Proof of Theorems 5b, 7b – push&pull informs almost all vertices fast in spite of edge deletions

Before we show the actual proof we will first present an informal argument that contains all relevant ideas and important observations. Let $\sqrt{\log n} \leq |I_t| \leq n / \log n$ and assume $q = 1$. In Section 3.2 we proved that for *push* the informed vertices nearly double in every round for an arbitrary expander sequence with edge deletions and an otherwise arbitrary set I_t . For *pull* this is not true; however, we proved in Section 3.1 that the number of edges between the informed and the uninformed vertices nearly doubles in every round. The first attempt towards the proof of Theorems b, b then seems obvious: one would try to show that either the vertices triple every round, or the edges do so, or for example that the product of the two quantities increases by a factor of 9. As it turns out, this is in general not the case; indeed, it is possible to choose an expander sequence, to delete edges such that each vertex keeps at least an $(1/2 + \varepsilon)$ -fraction of its neighbors, and to choose a (large) set of informed vertices I_t such that after one round whp either $|I_{t+1}| < c|I_t|$ or $e(I_{t+1}, U_{t+1}) < ce(I_t, U_t)$ or

$|I_{t+1}|e(I_{t+1}, U_{t+1}) < c^2|I_t|e(I_t, U_t)$ for some $c < 3$. On the other hand and although we have no explicit description of these “malicious” sets, it seems rather unlikely that such sets will occur several times during the execution of *push&pull*.

In order to show the claimed running time of *push&pull* we will impose some additional structure. Let $\varepsilon > 0$. In the subsequent exposition we assume that our graph G – obtained from an expander by deleting edges such that each vertex keeps at least an $(1/2 + \varepsilon)$ fraction of the edges – has a *very special* structure. In particular, we assume that there is a partition $\Pi = (V_i)_{i \in [k]}$ of the vertex set of G into a bounded number k of equal parts such that $E_G(V_i) = \emptyset$ for all $1 \leq i \leq k$ and such that the induced subgraph (V_i, V_j) looks like a random regular bipartite graph for all $1 \leq i < j \leq k$. Of course, not every relevant G admits such a partition; however, Szemerédi’s regularity lemma guarantees that every sufficiently large graph has a partition that is in a well-defined sense *almost* like the one described previously, and a substantial part of our proof is concerned with showing that being “almost special” does not hurt significantly.

Assuming that G is very special let us collect some easy facts. Denote the degree of $u \in V_i$ in the induced subgraph (V_i, V_j) with d_{ij} ; this immediately gives that $d_G(u) = \sum_{\ell=1}^k d_{i\ell}$, and note that $d_{ii} = 0$ as there are no edges in V_i . Moreover, regular bipartite random graphs satisfy an expander property, that is, for all $W_i \subseteq V_i, W_j \subseteq V_j, 1 \leq i < j \leq k$ we have

$$e(W_i, W_j) = d_{i,j}|W_i||W_j|/|V_j| + o(d_{i,j})|W_i| \approx |W_i||W_j|d_{ij}k/n$$

where we used that all $|V_i|$ ’s are of equal size. This is quite similar to the property that we used in our preceding analysis on expander sequences, see Lemma 16. As a pair in Π behaves like a bipartite expander sequence we can easily compute the expected number of informed vertices. We do so now for *pull*. Let $|I_{t+1}^{i,j}|$ be the number of vertices in V_i informed after round $t + 1$ by *pull* from vertices only in V_j and set $I_t^i := I_t \cap V_i, U_t^i := U_t \cap V_i \forall 1 \leq i \leq k$. Thus, as long as I_t^i is much smaller than V_i (and thus also $U_t^i \approx |V_i| = n/k$) we get

$$\mathbb{E}_t \left[|I_{t+1}^{(pull),i,j} \setminus I_t^i| \right] = \sum_{u \in U_t^i} \frac{|N(u) \cap I_t^j|}{d(u)} = \frac{e(U_t^i, I_t^j)}{\sum_{1 \leq \ell \leq k} d_{i\ell}} \approx \frac{d_{ij}}{\sum_{1 \leq \ell \leq k} d_{i\ell}} |I_t^j|.$$

A similar calculation, which we don’t perform in detail, yields for *push*

$$\mathbb{E}_t \left[|I_{t+1}^{(push),i,j} \setminus I_t^i| \right] \approx \frac{d_{ij}}{\sum_{1 \leq \ell \leq k} d_{\ell j}} |I_t^j|.$$

Moreover, as in previous proofs it turns out that the number of vertices informed simultaneously by *push* as well as *pull* is negligible. Thus we obtain that more or less

$$\mathbb{E}_t \left[|I_{t+1}^{(pp),i,j}| \right] \approx |I_t^i| + \left(\frac{d_{ij}}{\sum_{1 \leq \ell \leq k} d_{i\ell}} + \frac{d_{ij}}{\sum_{1 \leq \ell \leq k} d_{\ell j}} \right) |I_t^j|$$

and by linearity of expectation

$$\mathbb{E}_t \left[|I_{t+1}^{(pp),i}| \right] \approx |I_t^i| + \sum_{1 \leq j \leq k} \left(\frac{d_{ij}}{\sum_{1 \leq \ell \leq k} d_{i\ell}} + \frac{d_{ij}}{\sum_{1 \leq \ell \leq k} d_{\ell j}} \right) |I_t^j|.$$

Set $X_t = (|I_t^i|)_{i \in [k]}$ and $A = (A_{ij})_{1 \leq i, j \leq k}$, the matrix with entries

$$A_{ij} = \frac{d_{ij}}{\sum_{1 \leq \ell \leq k} d_{i\ell}} + \frac{d_{ij}}{\sum_{1 \leq \ell \leq k} d_{\ell j}} \quad \text{for } 1 \leq i \neq j \leq k$$

36:12 Robustness of Randomized Rumour Spreading

and $A_{ii} = 1$ for $1 \leq i \leq k$. With this notation we obtain the recursive relation

$$\mathbb{E}_t[X_{t+1}] \approx A \cdot X_t, \quad (5)$$

that is, we may expect that $X_t \approx \mathbb{E}_t[X_t] \approx A^t X_0$. If we then denote by λ_{\max} the greatest eigenvalue of A , then we obtain that in leading order

$$|I_t| \approx \lambda_{\max}^t.$$

Our aim is to show that *push&pull* is (at least) as fast as on the complete graph, that is, $|I_t| \lesssim 3^t$, and so we take a closer look at the eigenvalues of A . By construction A is symmetric, so that the largest eigenvalue equals $\sup_{\|x\|=1} \|x^T A x\|$, and the simple choice $x = k^{-1/2} \mathbf{1}$ yields

$$\begin{aligned} \lambda_{\max} &\geq \frac{\sum_{(i,j)} A_{i,j}}{k} \\ &= \frac{\sum_{j=1}^k 1 + \sum_{i=1}^k \sum_{j=1}^k d_{ij} / \left(\sum_{\ell=1}^k d_{i\ell} \right) + \sum_{j=1}^k \sum_{i=1}^k d_{ij} / \left(\sum_{\ell=1}^k d_{\ell j} \right)}{k} = 3. \end{aligned}$$

This neat property leads us to the expected result $T_{pp}(G) = (1 + o(1)) \log_{\lambda_{\max}} n \leq (1 + o(1)) \log_3 n$, and it also completes the informal argument that justifies the claim made in Theorems 5b and 7b. In the unabridged version we turn this argument step by step into a formal proof by filling in all missing pieces.

3.4 Proof of Theorem 6b – edge deletions may slow down *push&pull*

For any $0 < \varepsilon < 1/2$, $q \in (0, 1)$ we consider a sequence of graphs $(G_n(\varepsilon))_{n \in \mathbb{N}} = ((V_n, E_n))_{n \in \mathbb{N}}$. Let $V_n = A_n \cup B_n$ with $A_n := \{1, \dots, \lfloor n/2 \rfloor\}$, $B_n := \{\lfloor n/2 \rfloor + 1, \dots, n\}$ and $\deg(v) = n - 1$ for all $v \in A_n$. Let the induced subgraph of B_n be a random graph in which each edge is included independently with probability $p = 1 - 2\varepsilon$. We know and it is easy to show, see for example [15, Section IV], that whp this subgraph is almost regular, i.e.,

$$d_{B_n}(v) = (1 + o(1))(1 - 2\varepsilon)n/2 \quad \text{for all } v \in B_n, \quad (6)$$

and is an expander, which means that for every $S_n \subseteq B_n$, $1 \leq |S_n| \leq n/4$ and $d_{B_n} := (1 - 2\varepsilon)n/2$ we have

$$e(S_n, B_n \setminus S_n) = (1 + o(1)) \frac{d_{B_n} |S_n| |B_n \setminus S_n|}{|B_n|} = (1 - 2\varepsilon + o(1)) |S_n| |B_n \setminus S_n|. \quad (7)$$

At first we give a statement that describes the expected number of informed vertices after performing one round of *push&pull*.

► **Lemma 24.** *Let $G_n(\varepsilon) = (A_n \cup B_n, E_n)$ be as above.*

(a) *Let $\sqrt{\log n} \leq |I_t| \leq n/\log n$ and set*

$$X_t = \left(|I_t^{(pp),(A)}|, |I_t^{(pp),(B)}| \right) := \left(|I_t^{(pp)} \cap A_n|, |I_t^{(pp)} \cap B_n| \right).$$

Then $\mathbb{E}_t[X_{t+1}] = (1 + o(1)) M X_t$, where

$$M = \begin{pmatrix} 1 + q & q(1 + \varepsilon/(2 - 2\varepsilon)) \\ q(1 + \varepsilon/(2 - 2\varepsilon)) & 1 + q(1 - 2\varepsilon/(2 - 2\varepsilon)) \end{pmatrix}.$$

(b) *Let $|U_t^{(pp)}| \leq n/\log n$. Then $\mathbb{E}_t[|U_{t+1}^{(pp)}|] \leq (1 + o(1)) e^{-q(1/2 + (1/2 - \varepsilon)/(1 - \varepsilon))} (1 - q) |U_t|$.*

► Remark 25. Let λ_{\max} be the greatest eigenvalue of M as defined in Lemma 24a. Then

$$\lambda_{\max} = 1 + 2q + (2q(\sqrt{(\varepsilon^2/2 - \varepsilon + 1)} - 1) + q\varepsilon)/(2 - 2\varepsilon) > 1 + 2q.$$

Next comes a lemma that bounds the runtime of *push&pull* on $G_n(\varepsilon)$. In particular, Lemma 26 a) and c) provide a lower bound on the runtime and Lemma 26 a), b) and d) provides an upper bound.

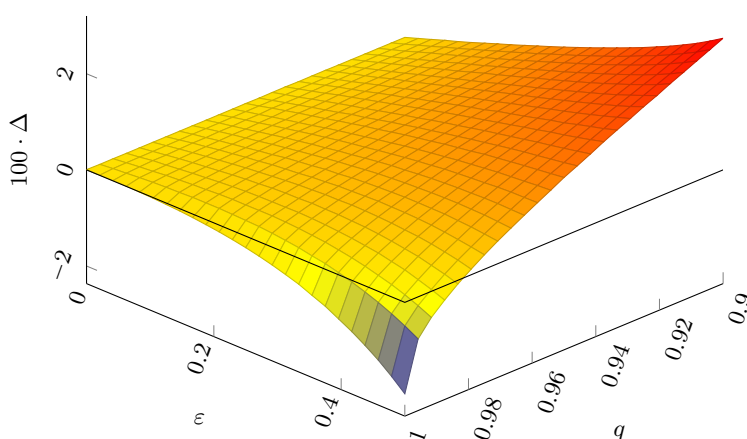
► **Lemma 26.** Let $I_t = I_t^{(pp)}$, $\varepsilon > 0$ and $\lambda = \lambda_{\max}(M)$ be the greatest eigenvalue of M as given in Lemma 24a. Consider $G_n(\varepsilon)$.

- (a) Let $\sqrt{\log n} \leq |I_t| \leq n/\log n$. Then there are $\tau_1, \tau_2 = \log_\lambda(n/|I_t|) + o(\log n)$ such that $|I_{t+\tau_1}| < n/\log n < |I_{t+\tau_2}|$.
- (b) Let $n/\log n \leq |I_t| \leq n - n/\log n$. Then there is $\tau = o(\log n)$ such that $|I_{t+\tau}| > n - n/\log n$.
- (c) Let $|I_t| \leq n/\log n$. Then there is $\tau \geq \log n / \log((1 - q)^{-1} \exp(q(1/2 + (1/2 - \varepsilon)/(1 - \varepsilon)))) - o(\log n)$ such that $|I_{t+\tau}| < n$.
- (d) Let $|I_t| \geq n - n/\log n$ and $q \in (0, 1)$. Then there is $\tau \leq \log n / \log((1 - q)^{-1} \exp(q(1/2 + (1/2 - \varepsilon)/(1 - \varepsilon)))) + o(\log n)$ such that $|I_{t+\tau}| = n$.

Lemma 26 gives that

$$T_{pp}(G_n(\varepsilon), q) = \log_\lambda n + \frac{1}{q(1 - 1.5\varepsilon)/(1 - \varepsilon) - \log(1 - q)} \log n + o(\log n)$$

where $\lambda = 1 + 2q + (2q(\sqrt{(\varepsilon^2/2 - \varepsilon + 1)} - 1) + q\varepsilon)/(2 - 2\varepsilon) > 1 + 2q$. To see whether *push&pull* actually slowed down (in terms of order $\log n$) one has to compare the runtime on this sequence of graphs to $c_{pp}(q) \log n$; the runtime on expander sequences. In the Figure 1 we can see that it slows down for nearly all values of ε and q in question; however, there are admissible values of ε and q such that the process even speeds up.



■ **Figure 1** Plotted values of Δ in $T_{pp}(G_n(\varepsilon), q) - c_{pp} \log n = \Delta \log n + o(\log n)$, for $0.9 < q < 1$ and $0 < \varepsilon < 1/2$.

References

- 1 Hüseyin Acan, Andrea Collecchio, Abbas Mehrabian, and Nick Wormald. On the push&pull protocol for rumour spreading. In *Extended Abstracts Summer 2015*, pages 3–10. Springer, 2017.
- 2 Omer Angel, Abbas Mehrabian, and Yuval Peres. The string of diamonds is tight for rumor spreading. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
- 3 Stéphane Boucheron, Gábor Lugosi, and Olivier Bousquet. Concentration inequalities. In *Advanced Lectures on Machine Learning*, pages 208–240. Springer, 2004.
- 4 Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE/ACM Transactions on Networking (TON)*, 14(SI):2508–2530, 2006.
- 5 Keren Censor-Hillel, Bernhard Haeupler, Jonathan Kelner, and Petar Maymounkov. Global computation in a poorly connected world: fast rumor spreading with no dependence on conductance. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 961–970. ACM, 2012.
- 6 Flavio Chierichetti, George Giakkoupis, Silvio Lattanzi, and Alessandro Panconesi. Rumor Spreading and Conductance. *Journal of the ACM (JACM)*, 65(4):17, 2018.
- 7 Sebastian Daum, Fabian Kuhn, and Yannic Maus. Rumor Spreading with Bounded In-Degree. In *Structural Information and Communication Complexity - 23rd International Colloquium, SIROCCO 2016, Helsinki, Finland, July 19-21, 2016, Revised Selected Papers*, pages 323–339, 2016. doi:10.1007/978-3-319-48314-6_21.
- 8 Domingos Dellamonica, Yoshiharu Kohayakawa, Martin Marcinişzyn, and Angelika Steger. On the Resilience of Long Cycles in Random Graphs. *Electr. J. Comb.*, 15(1), 2008. URL: http://www.combinatorics.org/Volume_15/Abstracts/v15i1r32.html.
- 9 Alan Demers, Dan Greene, Carl Houser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. *ACM SIGOPS Operating Systems Review*, 22(1):8–32, 1988.
- 10 Benjamin Doerr, Mahmoud Fouz, and Tobias Friedrich. Social networks spread rumors in sublogarithmic time. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 21–30. ACM, 2011.
- 11 Benjamin Doerr and Anatolii Kostrygin. Randomized Rumor Spreading Revisited. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 138:1–138:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.138.
- 12 Benjamin Doerr and Marvin Künnemann. Tight analysis of randomized rumor spreading in complete graphs. In *Proceedings of the Meeting on Analytic Algorithmics and Combinatorics*, pages 82–91. Society for Industrial and Applied Mathematics, 2014.
- 13 Robert Elsässer and Thomas Sauerwald. On the runtime and robustness of randomized broadcasting. *Theoretical Computer Science*, 410(36):3414–3427, 2009.
- 14 Uriel Feige, David Peleg, Prabhakar Raghavan, and Eli Upfal. Randomized broadcast in networks. *Random Structures & Algorithms*, 1(4):447–460, 1990.
- 15 Nikolaos Fountoulakis, Anna Huber, and Konstantinos Panagiotou. Reliable broadcasting in random networks and the effect of density. In *2010 Proceedings IEEE INFOCOM*, pages 1–9. IEEE, 2010.
- 16 Nikolaos Fountoulakis and Konstantinos Panagiotou. Rumor spreading on random regular graphs and expanders. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 560–573. Springer, 2010.
- 17 Nikolaos Fountoulakis, Konstantinos Panagiotou, and Thomas Sauerwald. Ultra-fast rumor spreading in social networks. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1642–1660. SIAM, 2012.

- 18 Tobias Friedrich, Thomas Sauerwald, and Alexandre Stauffer. Diameter and Broadcast Time of Random Geometric Graphs in Arbitrary Dimensions. *Algorithmica*, 67(1):65–88, September 2013. doi:10.1007/s00453-012-9710-y.
- 19 Alan M Frieze and Geoffrey R Grimmett. The shortest-path problem for graphs with random arc-lengths. *Discrete Applied Mathematics*, 10(1):57–77, 1985.
- 20 George Giakkoupis. Tight bounds for rumor spreading in graphs of a given conductance. In *28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011, March 10-12, 2011, Dortmund, Germany*, pages 57–68, 2011. doi:10.4230/LIPIcs.STACS.2011.57.
- 21 George Giakkoupis. Tight Bounds for Rumor Spreading with Vertex Expansion. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 801–815, 2014. doi:10.1137/1.9781611973402.59.
- 22 Bernhard Haeupler. Simple, fast and deterministic gossip and rumor spreading. *Journal of the ACM (JACM)*, 62(6):47, 2015.
- 23 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.
- 24 Richard M. Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking. Randomized Rumor Spreading. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 565–574, 2000. doi:10.1109/SFCS.2000.892324.
- 25 Konstantinos Panagiotou, Xavier Pérez-Giménez, Thomas Sauerwald, and He Sun. Randomized Rumour Spreading: The Effect of the Network Topology. *Combinatorics, Probability & Computing*, 24(2):457–479, 2015. doi:10.1017/S0963548314000194.
- 26 Konstantinos Panagiotou and Leo Speidel. Asynchronous rumor spreading on random graphs. *Algorithmica*, 78(3):968–989, 2017.
- 27 Vojtěch Rödl and Mathias Schacht. Regularity lemmas for graphs. In *Fete of combinatorics and computer science*, pages 287–325. Springer, 2010.
- 28 Benny Sudakov and Van H. Vu. Local resilience of graphs. *Random Struct. Algorithms*, 33(4):409–433, 2008. doi:10.1002/rsa.20235.

Structural Rounding: Approximation Algorithms for Graphs Near an Algorithmically Tractable Class

Erik D. Demaine

MIT, Cambridge, MA, USA
edemaine@mit.edu

Kyle Kloster

NC State University, Raleigh, NC, USA
kakloste@ncsu.edu

Quanquan C. Liu

MIT, Cambridge, MA, USA
quanquan@mit.edu

Ali Vakilian

MIT, Cambridge, MA, USA
vakilian@mit.edu

Timothy D. Goodrich

NC State University, Raleigh, NC, USA
tdgoodri@ncsu.edu

Brian Lavallee

NC State University, Raleigh, NC, USA
blavall@ncsu.edu

Blair D. Sullivan

NC State University, Raleigh, NC, USA
blair_sullivan@ncsu.edu

Andrew van der Poel

NC State University, Raleigh, NC, USA
ajvande4@ncsu.edu

Abstract

We develop a framework for generalizing approximation algorithms from the structural graph algorithm literature so that they apply to graphs somewhat close to that class (a scenario we expect is common when working with real-world networks) while still guaranteeing approximation ratios. The idea is to **edit** a given graph via vertex- or edge-deletions to put the graph into an algorithmically tractable class, apply known approximation algorithms for that class, and then **lift** the solution to apply to the original graph. We give a general characterization of when an optimization problem is amenable to this approach, and show that it includes many well-studied graph problems, such as INDEPENDENT SET, VERTEX COVER, FEEDBACK VERTEX SET, MINIMUM MAXIMAL MATCHING, CHROMATIC NUMBER, (ℓ) -DOMINATING SET, EDGE (ℓ) -DOMINATING SET, and CONNECTED DOMINATING SET.

To enable this framework, we develop new editing algorithms that find the approximately-fewest edits required to bring a given graph into one of a few important graph classes (in some cases these are bicriteria algorithms which simultaneously approximate both the number of editing operations and the target parameter of the family). For bounded degeneracy, we obtain an $O(r \log n)$ -approximation and a bicriteria $(4, 4)$ -approximation which also extends to a smoother bicriteria trade-off. For bounded treewidth, we obtain a bicriteria $(O(\log^{1.5} n), O(\sqrt{\log w}))$ -approximation, and for bounded pathwidth, we obtain a bicriteria $(O(\log^{1.5} n), O(\sqrt{\log w} \cdot \log n))$ -approximation. For treedepth 2 (related to bounded expansion), we obtain a 4-approximation. We also prove complementary hardness-of-approximation results assuming $P \neq NP$: in particular, these problems are all log-factor inapproximable, except the last which is not approximable below some constant factor 2 (assuming UGC).

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis

Keywords and phrases structural rounding, graph editing, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.37

Related Version <https://arxiv.org/abs/1806.02771>

Funding This research was supported in part by the Army Research Office under Grant Number W911NF-17-1-0271 to Blair D. Sullivan, the Gordon & Betty Moore Foundation's Data-Driven Discovery Initiative under Grant GBMF4560 to Blair D. Sullivan, as well as NSF grants CCF-1161626 and IIS-1546290 to Erik D. Demaine. Timothy D. Goodrich is partially supported by a National Defense Science & Engineering Graduate Program fellowship. Quanquan C. Liu is partially supported by a National Science Foundation Graduate Research Fellowship under grant 1122374.



© Erik D. Demaine, Timothy D. Goodrich, Kyle Kloster, Brian Lavallee, Quanquan C. Liu, Blair D. Sullivan, Ali Vakilian, and Andrew van der Poel;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 37; pp. 37:1–37:15



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

Acknowledgements We thank Abida Haque and Adam Hesterberg for helpful initial discussions, Nicole Wein for providing helpful references on bounded degeneracy problems, and Michael O’Brien for helpful comments on the manuscript.

1 Introduction

Network science has empirically established that real-world networks (social, biological, computer, etc.) exhibit sparse structure. Theoretical computer science has shown that graphs with certain structural properties enable significantly better approximation algorithms for hard problems. Unfortunately, the experimentally observed structures and the theoretically required structures are generally not the same: mathematical graph classes are rigidly defined, while real-world data is noisy and full of exceptions. This paper provides a framework for extending approximation guarantees from existing rigid classes to broader, more flexible graph families that are more likely to include real-world networks.

Specifically, we hypothesize that most real-world networks are in fact **small perturbations of graphs from a structural class**, that is, a family of graphs which adhere to some specified structure (e.g. treewidth at most w) [9, 29]. Intuitively, these perturbations may be exceptions caused by unusual/atypical behavior (e.g., weak links rarely expressing themselves), natural variation from an underlying model, or noise caused by measurement error or uncertainty. Formally, a graph is γ -close to a structural class \mathcal{C} , where $\gamma \in \mathbb{N}$, if some γ edits (e.g., vertex deletions, edge deletions, or edge contractions) bring the graph into class \mathcal{C} ¹. (Other papers call this the “noisy setting” [44, 12, 4].)

Our goal is to extend existing approximation algorithms for a structural class \mathcal{C} to apply more broadly to graphs γ -close to \mathcal{C} . To achieve this goal, we need two algorithmic ingredients:

1. **Editing algorithms.** Given a graph G that is γ -close to a structural class \mathcal{C} , find a sequence of $f(\gamma)$ edits that result in a member of \mathcal{C} . When the structural class is parameterized (e.g., treewidth $\leq w$), we may also approximate those parameters.
2. **Structural rounding algorithms.** Develop approximation algorithms for optimization problems on graphs γ -close to a structural class \mathcal{C} by converting ρ -approximate solutions on an edited graph in class \mathcal{C} into $g(\rho, \gamma)$ -approximate solutions on the original graph.

1.1 Our Results: Structural Rounding

In Section 4, we present a general metatheorem giving sufficient conditions for an optimization problem to be amenable to the structural rounding framework. Specifically, if a problem Π has an approximation algorithm in structural class \mathcal{C} , the problem and its solutions are “stable” under an edit operation, and there is an α -approximate algorithm for editing to \mathcal{C} , then we get an approximation algorithm for solving Π on graphs γ -close to \mathcal{C} . The new approximation algorithm incurs an additive error of $O(\gamma)$, so we preserve PTAS-like $(1 + \varepsilon)$ approximation factors provided $\gamma \leq \delta \text{OPT}_{\Pi}$ for a suitable constant $\delta = \delta(\varepsilon, \alpha) > 0$. Our metatheorem generalizes previous analysis of two specific problems [4].

¹ Note that the number of these edits could be super-constant. The number of edits could be as large as $O(m + n)$, the size of the graph.

For example, we obtain $(1 + O(\delta \log^{1.5} n))$ -approximation algorithms for VERTEX COVER, FEEDBACK VERTEX SET, MINIMUM MAXIMAL MATCHING, and CHROMATIC NUMBER on graphs $(\delta \cdot \text{OPT}_{\Pi}(G))$ -close to having treewidth w via vertex deletions (generalizing exact algorithms for bounded treewidth graphs); and we obtain a $(1 - 4\delta)/(4r + 1)$ -approximation algorithm for INDEPENDENT SET on graphs $(\delta \cdot \text{OPT}_{\Pi}(G))$ -close to having degeneracy r (generalizing a $1/r$ -approximation for degeneracy- r graphs). These results use our new algorithms for editing to treewidth- w and degeneracy- r graph classes as summarized next.

1.2 Our Results: Editing

We develop editing approximation algorithms and/or hardness-of-approximation results for six well-studied graph classes: bounded degeneracy, bounded treewidth and pathwidth, bounded clique number, bounded treedepth, bounded weak c -coloring number, and bounded degree. Refer to the full version of this paper ([18]) for details about these classes. Table 1 summarizes our results for the bounded degeneracy and bounded treewidth classes which we use in our structural rounding framework to find approximate solutions for some classic problems. Refer to the full version of this paper ([18]) for an overview of our results for the rest of the aforementioned graph classes.

■ **Table 1** Summary of results for $(\mathcal{C}_{\lambda}, \psi)$ -EDIT problems, i.e. finding the minimum number of ψ -edits needed to obtain a graph in class \mathcal{C}_{λ} (including abbreviations and standard parameter notation). For each combination we give a shorthand problem name in bold (e.g. r -DE-V). “Approx.” denotes a polynomial-time approximation or bicriteria approximation algorithm (see Section 3); “inapprox.” denotes inapproximability assuming $P \neq NP$ unless otherwise specified.

Graph Family \mathcal{C}_{λ}	Edit Operation ψ	
	Vertex Deletion	Edge Deletion
Bounded Degeneracy (r)	<p>r-DE-V</p> <p>$O(r \log n)$-approx.</p> <p>$(\frac{4m - \beta rn}{m - rn}, \beta)$-approx.</p> <p>$(\frac{1}{\varepsilon}, \frac{4}{1 - 2\varepsilon})$-approx. ($\varepsilon < 1/2$)</p> <p>$o(\log(n/r))$-inapprox.</p>	<p>r-DE-E</p> <p>$O(r \log n)$-approx.</p> <p>–</p> <p>$(\frac{1}{\varepsilon}, \frac{4}{1 - \varepsilon})$-approx. ($\varepsilon < 1$)</p> <p>$o(\log(n/r))$-inapprox.</p>
Bounded Treewidth (w)	<p>w-TW-V</p> <p>$(O(\log^{1.5} n), O(\sqrt{\log w}))$-approx.</p> <p>$o(\log n)$-inapprox. for $w \in \Omega(n^{1/2})$</p>	<p>w-TW-E</p> <p>$(O(\log n \log \log n), O(\log w))$-approx. [4]</p> <p>–</p>

1.3 Related Work

Editing to approximate optimization problems. The most closely related results are in the “noisy setting” introduced by Magen and Moharrami [44], which imagines that the “true” graph lies in the structural graph class that we want, and any extra edges observed in the given graph are “noise.” In this model, Magen and Moharrami [44] developed a PTAS for estimating the *size* of INDEPENDENT SET (IS) in graphs that are δn edits away from a minor-closed graph family (for sufficiently small values of δ). However, they provide no method for actually finding a solution set of vertices that achieves this approximation [44]. Later, Chan and Har-Peled [12] developed a PTAS that returns a $(1 + \varepsilon)$ -approximation to IS in noisy planar graphs. More recently, Bansal et al. [4] developed an LP-based approach for noisy minor-closed IS whose runtime and approximation factor achieve better dependence on δ but only for edge edits. Moreover, they provide a similar guarantee for noisy Max k -CSPs also for edge edits [4]. Their approximation analysis resembles our general analysis of the structural rounding framework.

Another related set of approximation algorithms near a graph class are **parameterized approximations**, meaning that they run in polynomial time only when the number of edits is very small (constant or logarithmic input size). This research direction was initiated by Cai [11]; see the survey and results of Marx [46, Section 3.2] and e.g. [30, 45]. An example of one such result is a $\frac{7}{3}$ -approximation algorithm to Chromatic Number in graphs that become planar after γ vertex edits, with a running time of $f(\gamma) \cdot O(n^2)$, where $f(\gamma)$ is at least $2^{2^{2^{\Omega(\gamma)}}}$ (from the use of Courcelle’s Theorem), limiting its polynomial-time use to when the number of edits satisfies $\gamma = O(\lg \lg \lg n)$. In contrast, our algorithms allow up to δOPT_Π edits.

Editing algorithms. Editing graphs into a desired graph class is an active field of research and has various applications outside of graph theory, including computer vision and pattern matching [28]. In general, the editing problem is to delete a minimum set X of vertices (or edges) in an input graph G such that the result $G[V \setminus X]$ has a specific property. Previous work studied this problem from the perspective of identifying the maximum induced subgraph of G that satisfies a desired “nontrivial, hereditary” property [39, 41, 42, 56]. A graph property π is nontrivial if and only if infinitely many graphs satisfy π and infinitely many do not, and π is hereditary if G satisfying π implies that every induced subgraph of G satisfies π . The vertex-deletion problem for any nontrivial, hereditary property has been shown to be NP-complete [42] and even requires exponential time to solve, assuming the ETH [37]. Approximation algorithms for such problems have also been studied in this domain [27, 43, 52], but in general this problem requires additional restrictions on the input graph and/or output graph properties in order to develop fast algorithms [17, 20, 22, 33, 38, 48, 49, 55].

Much past work on editing is on parameterized algorithms. For example, Dabrowski et al. [17] found that editing a graph to have a given degree sequence is W[1]-complete, but if one additionally requires that the final graph be planar, the problem becomes Fixed Parameter Tractable (FPT). Mathieson [48] showed that editing to degeneracy d is W[P]-hard (even if the original graph has degeneracy $d + 1$ or maximum degree $2d + 1$), but suggests that classes which offer a balance between the overly rigid restrictions of bounded degree and the overly global condition of bounded degeneracy (e.g., structurally sparse classes such as H -minor-free and bounded expansion [51]) may still be FPT. Some positive results on the parameterized complexity of editing to classes can be found in Drange’s 2015 PhD thesis [20]; in particular, the results mentioned include parameterized algorithms for a variety of NP-complete editing problems such as editing to threshold and chain graphs [22], star forests [22], multipartite cluster graphs [25], and \mathcal{H} -free graphs given finite \mathcal{H} and bounded indegree [21].

Our approach differs from this prior work in that we focus on approximations of edit distance that are **polynomial-time approximation algorithms**. There are previous results about approximate edit distance by Fomin et al. [26] and, in a very recent result regarding approximate edit distance to bounded treewidth graphs, by Gupta et al. [31]. Fomin et al. [26] provided two types of algorithms for vertex editing to planar \mathcal{F} -minor-free graphs: a randomized algorithm that runs in $O(f(\mathcal{F}) \cdot mn)$ time with an approximation constant $c_{\mathcal{F}}$ that depends on \mathcal{F} , as well as a fixed-parameter algorithm parameterized by the size of the edit set whose running time thus has an exponential dependence on the size of this edit set. Agrawal et al. [1] recently provided a $O(\log^{1.5} n)$ -approximation via a parameterized algorithm for the WEIGHTED \mathcal{F} VERTEX DELETION problem (among some other problems) where \mathcal{F} is a minor-closed family excluding at least one planar graph.

Gupta et al. [31] strengthen the results in [26] but only in the context of **parameterized approximation algorithms**. Namely, they give a deterministic fixed-parameter algorithm for PLANAR \mathcal{F} -DELETION that runs in $f(\mathcal{F}) \cdot n \log n + n^{O(1)}$ time and an

$O(\log k)$ -approximation where k is the maximum number of vertices in any planar graph in \mathcal{F} ; this implies a fixed-parameter $O(\log w)$ -approximation algorithm with running time $2^{O(w^2 \log w)} \cdot n \log n + n^{O(1)}$ for w -TW-V and w -PW-V. They also show that w -TW-E and w -PW-E have parameterized algorithms that give an absolute constant factor approximation but with running times parameterized by w and the maximum degree of the graph [31]. Finally, they show that when \mathcal{F} is the set of all connected graphs with three vertices, deleting the minimum number of edges to exclude \mathcal{F} as a subgraph, minor, or immersion is APX-hard for bounded degree graphs [31]. Again, these running times are weaker than our results, which give bicriteria approximation algorithms that are polynomial without any parameterization on the treewidth or pathwidth of the target graphs. Here, bicriteria relates to the number of editing operations and the target parameter.

In a similar regime, Bansal et al. [4] studied w -TW-E (which implies an algorithm for w -PW-E) and designed an LP-based bicriteria approximation for this problem. For a slightly different set of problems in which the goal is to exclude a single graph H of size k as a subgraph (H -VERTEX-DELETION), there exists a simple k -approximation algorithm. On the hardness side, Guruswami and Lee [32] proved that whenever H is 2-vertex-connected, it is NP-hard to approximate H -VERTEX-DELETION within a factor of $(|V(H)| - 1 - \varepsilon)$ for any $\varepsilon > 0$ ($|V(H)| - \varepsilon$ assuming UGC). Moreover, when H is a star or simple path with k vertices, $O(\log k)$ -approximation algorithms with running time $2^{O(k^3 \log k)} \cdot n^{O(1)}$ are known [32, 40].

An important special case of the problem of editing graphs into a desired class is the *minimum planarization* problem, in which the target class is planar graphs, and the related application is approximating the well-known *crossing number* problem [15]. Refer to [7, 13, 14, 34, 36, 35, 47, 54] for the recent developments on minimum planarization and crossing number.

2 Techniques

This section summarizes the main techniques, ideas, and contributions in the paper.

2.1 Structural Rounding Framework

The main contribution of our structural rounding framework (Section 4) is establishing the right definitions that make for a broadly applicable framework with precise approximation guarantees. Our framework supports arbitrary graph edit operations and both minimization and maximization problems, provided they jointly satisfy two properties: a combinatorial property called “stability” and an algorithmic property called “structural lifting”. Roughly, these properties bound the amount of change that OPT can undergo from each edit operation, but they are also parameterized to enable us to derive tighter bounds when the problem has additional structure. With the right definitions in place, the framework is simple: edit to the target class, apply an existing approximation algorithm, and lift.

The rest of Section 4 shows that this framework applies to many different graph optimization problems. In particular, we verify the stability and structural lifting properties, and combine all the necessary pieces, including our editing algorithms from Section 5 and existing approximation algorithms for structural graph classes. We summarize all of these results in Table 2 and formally define the framework in Section 4.1.

■ **Table 2** Problems for which structural rounding (Theorem 4.4) results in approximation algorithms for graphs near the structural class \mathcal{C} , where the problem has a $\rho(\lambda)$ -approximation algorithm. We also give the associated stability (c') and lifting (c) constants, which are class-independent. The last column shows the running time of the $\rho(\lambda)$ -approximation algorithm for each problem provided an input graph from class \mathcal{C}_λ . We remark that vertex^* is used to emphasize the rounding process has to pick the set of annotated vertices in the edited set carefully to achieve the associated stability and lifting constants. We provide precise problem statements in the full version of this paper ([18]).

Problem	Edit ψ	c'	c	Class \mathcal{C}_λ	$\rho(\lambda)$	runtime
INDEPENDENT SET (IS)	vertex del.	1	0	degeneracy r	$\frac{1}{r+1}$	polytime
ANNOTATED DOMINATING SET (ADS)	vertex^* del.	0	1	degeneracy r	$O(r)$	polytime [5] ^a
INDEPENDENT SET (IS)	vertex del.	1	0	treewidth w	1	$O(2^w n)$ [2]
ANNOTATED DOMINATING SET (ADS)	vertex^* del.	0	1	treewidth w	1	$O(3^w n)$
ANNOTATED (ℓ -)DOMINATING SET (ADS)	vertex^* del.	0	1	treewidth w	1	$O((2\ell + 1)^w n)$ [10]
CONNECTED DOMINATING SET (CDS)	vertex^* del.	0	3	treewidth w	1	$O(n^w)^b$
VERTEX COVER (VC)	vertex del.	0	1	treewidth w	1	$O(2^w n)$ [2]
FEEDBACK VERTEX SET (FVS)	vertex del.	0	1	treewidth w	1	$2^{O(w)} n^{O(1)}$ [16]
MINIMUM MAXIMAL MATCHING (MMM)	vertex del.	0	1	treewidth w	1	$O(3^w n)^c$
CHROMATIC NUMBER (CRN)	vertex del.	0	1	treewidth w	1	$w^{O(w)} n^{O(1)}$
INDEPENDENT SET (IS)	edge del.	0	1	degeneracy r	$\frac{1}{r+1}$	polytime
DOMINATING SET (DS)	edge del.	1	0	degeneracy r	$O(r)$	polytime [5]
(ℓ -)DOMINATING SET (DS)	edge del.	1	0	treewidth w	1	$O((2\ell + 1)^w n)$ [10]
EDGE (ℓ -)DOMINATING SET (EDS)	edge del.	1	1	treewidth w	1	$O((2\ell + 1)^w n)$ [10]
MAX-CUT (MC)	edge del.	1	0	treewidth w	1	$O(2^w n)$ [19]

^a The approximation algorithm of [5] is analyzed only for DS; however, it is straightforward to show that the same algorithm achieves $O(r)$ -approximation for ADS as well.

^b Our rounding framework needs to solve an annotated version of CDS which can be solved in $O(n^w)$ by modifying the $O(w^w n)$ dynamic-programming approach of DS.

^c The same dynamic-programming approach of DS can be modified to solve ADS and MMM in $O(3^w n)$.

2.2 Editing to Bounded Degeneracy and Degree

We first present a $O(r \log n)$ -approximation algorithm for finding the fewest vertex or edge deletions to reduce the **degeneracy** to a target threshold r . The algorithm is a greedy algorithm over a type of **min-degree ordering** computed via the classic algorithm for finding the degeneracy of a graph G given by Matula and Beck [50]. In addition, we present two constant-factor bicriteria approximation algorithms for the same editing problem to degeneracy r . We provide a summary of the techniques used to obtain our results here; refer to the full version of this paper to see a detailed description of our techniques ([18]). The first approach uses the local ratio technique by Bar-Yehuda et al. [6] to establish that good-enough local choices result in a guaranteed approximation. The second approach is based on rounding a linear-programming relaxation of an integer linear program and works even when the input graph is **weighted** (both vertices and edges are weighted) and the goal is to minimize the total weight of the edit set.

On the lower bound side, we show $o(\log(n/r))$ -approximation is impossible for vertex or edge edits when we forbid bicriteria approximation, i.e., when we must match the target degeneracy r exactly. This result is based on a reduction from SET COVER. A similar reduction proves $o(\log d)$ -inapproximability of editing to maximum degree d , which proves tightness (up to constant factors) of a known $O(\log d)$ -approximation algorithm [23].

2.3 Editing to Bounded Treewidth

We present a bicriteria approximation algorithm in the full version ([18]) for finding the fewest vertex edits to reduce the **treewidth** to a target threshold w . Our approach builds on the deep separator structure inherent in treewidth. We combine ideas from Bodlaender’s $O(\log n)$ -approximation algorithm for treewidth with Feige et al.’s $O(\sqrt{\log w})$ -approximation algorithm for vertex separators [24] (where w is the target treewidth). In the end, we obtain a bicriteria $(O(\log^{1.5} n), O(\sqrt{\log w}))$ -approximation that runs in polynomial time on all graphs (in contrast to many previous treewidth algorithms). The tree decompositions that we generate are guaranteed to have $O(\log n)$ height. As a result, we also show a bicriteria $(O(\log^{1.5} n), O(\sqrt{\log w} \cdot \log n))$ -approximation result for pathwidth, based on the fact that the **pathwidth** is at most the width times the height of a tree decomposition.

On the lower bound side, we prove a $o(\log w)$ -inapproximability result by another reduction from SET COVER. By a small modification, this lower bound also applies to editing to **bounded clique number**.

3 Preliminaries

This section defines several standard notions and graph classes, and is probably best used as a reference. The one exception is Section 3.1, which formally defines the graph-class editing problem $(\mathcal{C}_\lambda, \psi)$ -EDIT introduced in this paper.

Graph notation. We consider finite, loopless, simple graphs. Unless otherwise specified, we assume that graphs are undirected and unweighted. We denote a graph by $G = (V, E)$, and set $n = |V|$, $m = |E|$. Given $G = (V, E)$ and two vertices $u, v \in V$ we denote edges by $e(u, v)$ or (u, v) . We write $N(v) = \{u \mid (u, v) \in E\}$ for the set of neighbors of a vertex v ; the degree of v is $\deg(v) = |N(v)|$. In digraphs, in-neighbors and out-neighbors of a vertex v are defined using edges of the form (u, v) and (v, u) , respectively, and we denote in- and out-degree by $\deg^-(v)$, $\deg^+(v)$, respectively. For the maximum degree of G we use $\Delta(G)$, or just Δ if context is clear. The clique number of G , denoted $\omega(G)$, is the size of the largest clique in G . Given some subset E' of the edges in G , we define $G[E']$ to be the subgraph of G induced on the edge set E' . Note that if every edge adjacent to some vertex v is in $E \setminus E'$, then v does not appear in the vertex set of $G[E']$.

We present below our definitions of editing problems that we consider in this paper. Please refer to the full version of this paper ([18]) for complete definitions of the structural graph classes, hardness reduction techniques, and hard optimization problems for which we provide approximation algorithms.

3.1 Editing Problems

This paper is concerned with algorithms that edit graphs into a desired structural class, while guaranteeing an approximation ratio on the size of the edit set. Besides its own importance, editing graphs into structural classes plays a key role in our structural rounding framework for approximating optimization problems on graphs that are “close” to structural graph classes (see Section 4). The basic editing problem is defined as follows relative to an edit operation ψ such as vertex deletion, edge deletion, or edge contraction:

(\mathcal{C}, ψ) -EDIT parametrised by

Input: An input graph $G = (V, E)$, family \mathcal{C} of graphs, edit operation ψ
Problem: Find k edits $\psi_1, \psi_2, \dots, \psi_k$ such that $\psi_k \circ \psi_{k-1} \circ \dots \circ \psi_2 \circ \psi_1(G) \in \mathcal{C}$.
Objective: Minimize k

We can also loosen the graph class we are aiming for, and approximate the parameter value λ for the family \mathcal{C}_λ . Thus we obtain a **bicriteria problem** which can be formalized as follows:

$(\mathcal{C}_\lambda, \psi)$ -EDIT parametrised by

Input: An input graph $G = (V, E)$, parameterized family \mathcal{C}_λ of graphs, a target parameter value λ^* , edit operation ψ
Problem: Find k edits $\psi_1, \psi_2, \dots, \psi_k$ such that $\psi_k \circ \psi_{k-1} \circ \dots \circ \psi_2 \circ \psi_1(G) \in \mathcal{C}_\lambda$ where $\lambda \geq \lambda^*$.
Objective: Minimize k .

► **Definition 3.1.** An algorithm for $(\mathcal{C}_\lambda, \psi)$ -EDIT is a **(bicriteria) (α, β) -approximation** if it guarantees that the number of edits is at most α times the optimal number of edits into \mathcal{C}_λ , and that $\lambda \leq \beta \cdot \lambda^*$.

See the full version of this paper for a complete list of the problems considered, along with their abbreviations. Recall that $\rho(\lambda)$ is the approximation factor for a problem in class \mathcal{C} . We assume that $\mathcal{C}_i \subseteq \mathcal{C}_j$ for $i \leq j$, or equivalently, that $\rho(\lambda)$ is monotonically increasing in λ .

4 Structural Rounding

In this section, we show how approximation algorithms for a structural graph class can be extended to graphs that are near that class, provided we can find a certificate of being near the class. These results thus motivate our results in later sections about editing to structural graph classes. Our general approach, which we call **structural rounding**, is to apply existing approximation algorithms on the edited (“rounded”) graph in the class, then “lift” that solution to solve the original graph, while bounding the loss in solution quality throughout.

4.1 General Framework

First we define our notion of “closeness” in terms of a general family ψ of allowable graph edit operations (e.g., vertex deletion, edge deletion, edge contraction):

► **Definition 4.1.** A graph G' is **γ -editable** from a graph G under edit operation ψ if there is a sequence of $k \leq \gamma$ edits $\psi_1, \psi_2, \dots, \psi_k$ of type ψ such that $G' = \psi_k \circ \psi_{k-1} \circ \dots \circ \psi_2 \circ \psi_1(G)$. A graph G is **γ -close** to a graph class \mathcal{C} under ψ if some $G' \in \mathcal{C}$ is γ -editable from G under ψ .

To transform an approximation algorithm for a graph class \mathcal{C} into an approximation algorithm for graphs γ -close to \mathcal{C} , we will need two properties relating the optimization problem and the type of edits:

► **Definition 4.2.** A graph minimization (resp. maximization) problem Π is **stable** under an edit operation ψ with constant c' if $\text{OPT}_\Pi(G') \leq \text{OPT}_\Pi(G) + c'\gamma$ (resp. $\text{OPT}_\Pi(G') \geq \text{OPT}_\Pi(G) - c'\gamma$) for any graph G' that is γ -editable from G under ψ . In the special case where $c' = 0$, we call Π **closed** under ψ . When ψ is vertex deletion, closure is equivalent to the graph class defined by $\text{OPT}_\Pi(G) \leq \lambda$ (resp. $\text{OPT}_\Pi(G) \geq \lambda$) being **hereditary**; we also call Π **hereditary**.

► **Definition 4.3.** A minimization (resp. maximization) problem Π can be **structurally lifted** with respect to an edit operation ψ with constant c if, given any graph G' that is γ -editable from G under ψ , and given the corresponding edit sequence $\psi_1, \psi_2, \dots, \psi_k$ with $k \leq \gamma$, a solution S' for G' can be converted in polynomial time to a solution S for G such that $\text{Cost}_\Pi(S) \leq \text{Cost}_\Pi(S') + c \cdot k$ (resp. $\text{Cost}_\Pi(S) \geq \text{Cost}_\Pi(S') - c \cdot k$).

Now we can state the main result of structural rounding:

► **Theorem 4.4 (Structural Rounding Approximation).** Let Π be a minimization (resp. maximization) problem that is stable under the edit operation ψ with constant c' and that can be structurally lifted with respect to ψ with constant c . If Π has a polynomial-time $\rho(\lambda)$ -approximation algorithm in the graph class \mathcal{C}_λ , and $(\mathcal{C}_\lambda, \psi)$ -EDIT has a polynomial-time (α, β) -approximation algorithm, then there is a polynomial-time $((1 + c'\alpha\delta) \cdot \rho(\beta\lambda) + c\alpha\delta)$ -approximation (resp. $((1 - c'\alpha\delta) \cdot \rho(\beta\lambda) - c\alpha\delta)$ -approximation) algorithm for Π on any graph that is $(\delta \cdot \text{OPT}_\Pi(G))$ -close to the class \mathcal{C}_λ .

Proof. We write $\text{OPT}(G)$ for $\text{OPT}_\Pi(G)$. Let G be a graph that is $(\delta \cdot \text{OPT}(G))$ -close to the class \mathcal{C}_λ . By Definition 3.1, the polynomial-time (α, β) -approximation algorithm finds edit operations $\psi_1, \psi_2, \dots, \psi_k$ where $k \leq \alpha\delta \cdot \text{OPT}(G)$ such that $G' = \psi_k \circ \psi_{k-1} \circ \dots \circ \psi_2 \circ \psi_1(G) \in \mathcal{C}_{\beta\lambda}$. Let $\rho = \rho(\beta\lambda)$ be the approximation factor we can attain on the graph $G' \in \mathcal{C}_{\beta\lambda}$.

We prove the case when Π is a minimization problem. The proof of the maximization case can be found in the full version of this paper. Because Π has a ρ -approximation in $\mathcal{C}_{\beta\lambda}$ (where $\rho > 1$), we can obtain a solution S' with cost at most $\rho \cdot \text{OPT}(G')$ in polynomial time. Applying structural lifting (Definition 4.3), we can use S' to obtain a solution S for G with $\text{Cost}(S) \leq \text{Cost}(S') + ck \leq \text{Cost}(S') + c\alpha\delta \cdot \text{OPT}(G)$ in polynomial time. Because Π is stable under ψ with constant c' ,

$$\text{OPT}(G') \leq \text{OPT}(G) + c'k \leq \text{OPT}(G) + c'\alpha\delta \cdot \text{OPT}(G) = (1 + c'\alpha\delta) \text{OPT}(G),$$

and we have

$$\text{Cost}(S) \leq \rho \cdot \text{OPT}(G') + c\alpha\delta \cdot \text{OPT}(G) = (\rho + \rho c'\alpha\delta + c\alpha\delta) \text{OPT}(G),$$

proving that we have a polynomial time $(\rho + (c + c'\rho)\alpha\delta)$ -approximation algorithm as required. ◀

To apply Theorem 4.4, we need four ingredients: (a) a proof that the problem of interest is stable under some edit operation (Definition 4.2); (b) a polynomial-time (α, β) -approximation algorithm for editing under this operation (Definition 3.1); (c) a structural lifting algorithm (Definition 4.3); and (d) an approximation algorithm for the target class \mathcal{C} .

In the remainder of this section, we show how this framework applies to many problems and graph classes, as summarized in Table 2 on page 6. Most of our approximation algorithms depend on our editing algorithms described in Section 5.

Structural rounding for annotated problems. We refer to graph optimization problems where the input consists of both a graph and subset of annotated vertices/edges as **annotated** problems. Hence, in our rounding framework, we have to carefully choose the set of annotated vertices/edges in the edited graph to guarantee small **lifting** and **stability** constants. To emphasize the difference compared to “standard” structural rounding, we denote the edit operations as vertex^* and edge^* in the annotated cases. Moreover, we show that we can further leverage the flexibility of annotated rounding to solve **non-annotated** problems that cannot normally be solved via structural rounding. In the full version of this paper ([18]), we consider applications of annotated rounding for both annotated problems such as ANNOTATED DOMINATING SET and non-annotated problems such as CONNECTED DOMINATING SET.

4.2 Applications: Vertex and Edge Deletions

For each problem, we show stability and structural liftability, and use these to conclude approximation algorithms. Using our structural rounding framework above, we obtain the following results on a broad set of problems for a number of different target classes. We point out that these problems are hard-to-solve on general graphs. Table 2 shows a summary of the set of problems we can obtain efficient approximation algorithms using structural rounding. The full version of this paper ([18]) contains the stability and structural liftability proofs used to obtain the corresponding results stated below.

We first use our structural rounding framework with vertex deletions to obtain the following approximation results.

- **Theorem 4.5.** *For graphs $(\delta \cdot \text{OPT}(G))$ -close to degeneracy r via vertex deletions,*
- INDEPENDENT SET has a $(1 - 4\delta)/(4r + 1)$ -approximation.
 - ANNOTATED DOMINATING SET has $O(r + \delta)$ -approximation.
- For graphs $(\delta \cdot \text{OPT}(G))$ -close to treewidth w via vertex deletions:*
- ANNOTATED (ℓ) -DOMINATING SET has a $(1 + O(\delta \log^{1.5} n))$ -approximation for the case $w\sqrt{\log w} = O(\log_\ell n)$.
 - INDEPENDENT SET has a $(1 - O(\delta \log^{1.5} n))$ -approximation when $w\sqrt{\log w} = O(\log n)$.
 - The problems VERTEX COVER, CHROMATIC NUMBER, and FEEDBACK VERTEX SET have $(1 + O(\delta \log^{1.5} n))$ -approximations when $w\sqrt{\log w} = O(\log n)$.
 - MINIMUM MAXIMAL MATCHING has a $(1 + O(\delta \log^{1.5} n))$ -approximation for the case $w \log^{1.5} w = O(\log n)$.
 - CONNECTED DOMINATING SET has a $(1 + O(\delta \log^{1.5} n))$ -approximation when $w = O(1)$.
- Finally, for graphs $(\delta \cdot \text{OPT}(G))$ -close to planar- H -minor-free via vertex deletions,*
- INDEPENDENT SET has a $(1 - c_H \delta)$ -approximation.
 - The problems VERTEX COVER, MINIMUM MAXIMAL MATCHING, CHROMATIC NUMBER, and FEEDBACK VERTEX SET have $(1 + c_H \delta)$ -approximations.

We now use our structural rounding framework with edge deletions to obtain the following approximation results.

- **Theorem 4.6.** *For graphs $(\delta \cdot \text{OPT}(G))$ -close to degeneracy r via edge deletions,*
- INDEPENDENT SET has a $(1/(3r + 1) - 3\delta)$ -approximation.
 - DOMINATING SET has an $O((1 + \delta)r)$ -approximation.
- For graphs $(\delta \cdot \text{OPT}(G))$ -close to treewidth w via edge deletions,*
- (ℓ) -DOMINATING SET and EDGE (ℓ) -DOMINATING SET have $(1 + O(\delta \log n \log \log n))$ -approximations when $w \log w = O(\log_\ell n)$.
 - MAX-CUT has a $(1 - O(\delta \log n \log \log n))$ -approximation when $w \log w = O(\log n)$.

Although we do not present any editing algorithms for edge contractions, we point out that such an editing algorithm would enable our framework to apply to additional problems such as (WEIGHTED) TSP TOUR, and to apply more efficiently to other problems such as DOMINATING SET (reducing c' from 1 to 0).

5 Editing Algorithms

5.1 Degeneracy: Greedy $O(r \log n)$ -Approximation

In this section, we give a polytime $O(\log n)$ -approximation for reducing the degeneracy of a graph by one using either vertex deletions or edge deletions. More specifically, given a graph $G = (V, E)$ with degeneracy $r + 1$, we produce an edit set X such that $G' = G \setminus X$ has degeneracy r and $|X|$ is at most $O(\log |V|)$ times the size of an optimal edit set. Note that this complements an $o(\log \frac{n}{r})$ -inapproximability result for the same problem.

In general, the algorithm works by computing a vertex ordering and greedily choosing an edit to perform based on that ordering. In our algorithm, we use the **min-degree ordering** of a graph. The **min-degree ordering** is computed via the classic greedy algorithm given by Matula and Beck [50] that computes the degeneracy of the graph by repeatedly removing a minimum degree vertex from the graph. The degeneracy of G , $\text{degen}(G)$, is the maximum degree of a vertex when it is removed. In the following proofs, we make use of the observation that given a min-degree ordering L of the vertices in $G = (V, E)$ and assuming the edges are oriented from smaller to larger indices in L , $\text{deg}^+(u) \leq \text{degen}(G)$ for any $u \in L$.

The first ordering L_0 is constructed by taking a min-degree ordering on the vertices of G where ties may be broken arbitrarily. Using L_0 , an edit is greedily chosen to be added to X . Each subsequent ordering L_i is constructed by taking a min-degree ordering on the vertices of $G \setminus X$ where ties are broken based on L_{i-1} . Specifically, if the vertices u and v have equal degree at the time of removal in the process of computing L_i , then $L_i(u) < L_i(v)$ if and only if $L_{i-1}(u) < L_{i-1}(v)$. The algorithm terminates when the min-degree ordering L_j produces a witness that the degeneracy of $G \setminus X$ is r .

In order to determine which edit to make at step i , the algorithm first computes the forward degree of each vertex u based on the ordering L_i (equivalently, $\text{deg}^+(u)$ when edges are oriented from smaller to larger index in L_i). Each vertex with forward degree $r + 1$ is marked, and similarly, each edge that has a marked left endpoint is also marked. The algorithm selects the edit that **resolves** the largest number of marked edges. We say that a marked edge is **resolved** if it will not be marked in the subsequent ordering L_{i+1} .

We observe that given an optimal edit set (of size k), removing the elements of the set in any order will resolve every marked edge after k rounds (assuming that at most one element from the optimal edit set is removed in each round). If it does not, then the final ordering L_k must have a vertex with forward degree $r + 1$, a contradiction. Let m_i be the number of marked edges based on the ordering L_i . We show that we can always resolve at least $\frac{m_i}{k}$ marked edges in each round, giving our desired approximation (all proofs of this section are deferred to the full version of this paper).

By repeatedly applying the $O(\log n)$ approximation given above, we can edit a graph with arbitrary degeneracy to the class of graphs with degeneracy r .

► **Theorem 5.1.** *There exists an $O(r \cdot \log n)$ -approximation for finding the minimum size edit set to reduce the degeneracy of a graph to r .*

5.2 Treewidth: bicriteria-approximation for vertex deletion

Our method for editing to bounded treewidth exploits the general recursive approach of the approximation algorithms for constructing a tree decomposition [3, 8, 24, 53]. Our algorithm iteratively subdivides the graph, considering $G[V_i]$ in iteration i . We first apply the result of [8, 24] (see Theorem 5.2) to determine if $G[V_i]$ has a tree decomposition with “small” width; if yes, the algorithm removes nothing and terminates. Otherwise, we compute an approximate vertex $(3/4)$ -separator S of $G[V_i]$, remove it from the graph, and recurse on the connected components of $G[V_i \setminus S]$. The full exposition of our results for editing to bounded treewidth and pathwidth graphs are given in the full version of this paper.

■ **Algorithm 1** Approximation for Vertex Editing to Bounded Treewidth Graphs.

```

1: procedure TREEWIDTHNODEEDIT( $G = (V, E), w$ )
2:    $t \leftarrow$  compute  $\text{tw}(G)$   $\triangleright$  refer to Theorem 5.2
3:   if  $t \leq 32c_1 \cdot w\sqrt{\log w}$  then
4:     return  $\emptyset$ 
5:   else
6:      $S \leftarrow$  compute a vertex  $(\frac{3}{4})$ -separator of  $G$  by invoking the algorithm of [24]
7:     let  $G[V_1], \dots, G[V_\ell]$  be the connected components of  $G[V \setminus S]$ .
8:     return  $(\bigcup_{i \leq \ell} \text{TREEWIDTHNODEEDIT}(G[V_i], w)) \cup S$ 
9:   end if
10: end procedure

```

► **Theorem 5.2** ([8, 24]). *There exists an algorithm that, given an input graph G , in polynomial time returns a tree decomposition of G of width at most $c_2 \cdot \text{tw}(G)\sqrt{\log \text{tw}(G)}$ and height $O(\log |V(G)|)$ for a sufficiently large constant c_2 .*

Next, we analyze the performance of Algorithm 1. Our approach relies on known results for **vertex c -separators**, structures which are used extensively in many other algorithms for finding an approximate tree decomposition.

► **Definition 5.3.** *For a subset of vertices W , a set of vertices $S \subseteq V(G)$ is a **vertex c -separator** of W in G if each component of $G[V \setminus S]$ contains at most $c|W|$ vertices of W . The minimum sized vertex c -separator of a graph is a separator with size k , denoted $\text{sep}_c(G)$, where k is the minimum integer such that for any subset $W \subseteq V$ there exists a vertex c -separator of W in G of size k .*

► **Theorem 5.4.** *Algorithm 1 removes at most $O(\log^{1.5} n) \text{OPT}_{w\text{-TW-V}}(G)$ vertices from any n -vertex graph G . The treewidth of the subgraph of G returned by Algorithm 1 is $O(w \cdot \sqrt{\log w})$.*

References


- 1 Akanksha Agrawal, Daniel Lokshtanov, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Polylogarithmic Approximation Algorithms for Weighted-F-Deletion Problems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, pages 1:1–1:15, 2018. doi:10.4230/LIPIcs.APPROX-RANDOM.2018.1.
- 2 Jochen Alber and Rolf Niedermeier. Improved Tree Decomposition Based Algorithms for Domination-like Problems. In *Proceedings of the 5th Latin American Symposium on Theoretical Informatics*, pages 613–627. Springer, 2001.
- 3 Eyal Amir. Approximation Algorithms for Treewidth. *Algorithmica*, 56(4):448–479, 2010.

- 4 Nikhil Bansal, Daniel Reichman, and Seeun William Umboh. LP-based robust algorithms for noisy minor-free and bounded treewidth graphs. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1964–1979. Society for Industrial and Applied Mathematics, 2017.
- 5 Nikhil Bansal and Seeun William Umboh. Tight approximation bounds for dominating set on graphs of bounded arboricity. *Information Processing Letters*, 122:21–24, 2017.
- 6 Reuven Bar-Yehuda, Keren Bendel, Ari Freund, and Dror Rawitz. Local ratio: A unified framework for approximation algorithms. *ACM Computing Surveys*, 36(4):422–463, 2004.
- 7 Therese Biedl, Markus Chimani, Martin Derka, and Petra Mutzel. Crossing number for graphs with bounded pathwidth. In *28th International Symposium on Algorithms and Computation, (ISAAC)*, pages 13:1–13:13, Phuket, Thailand, December 2017.
- 8 Hans L. Bodlaender, John R. Gilbert, Hjalmtyr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995.
- 9 Geoff Boeing. Planarity and Street Network Representation in Urban Form Analysis. *arXiv preprint arXiv:1806.01805*, 2018. [arXiv:1806.01805](https://arxiv.org/abs/1806.01805).
- 10 Glencora Borradaile and Hung Le. Optimal Dynamic Program for r -Domination Problems over Tree Decompositions. In *Proceedings of the 11th International Symposium on Parameterized and Exact Computation*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:23, 2017.
- 11 Leizhen Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3):415–429, 2003.
- 12 Timothy M Chan and Sarel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012.
- 13 Chandra Chekuri and Anastasios Sidiropoulos. Approximation algorithms for Euler genus and related problems. In *Proceedings of the IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 167–176. IEEE, 2013.
- 14 Julia Chuzhoy. An algorithm for the graph crossing number problem. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, pages 303–312, 2011.
- 15 Julia Chuzhoy, Yury Makarychev, and Anastasios Sidiropoulos. On graph crossing number and edge planarization. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1050–1069, 2011.
- 16 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Joham MM van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Foundations of computer science (focs), 2011 ieee 52nd annual symposium on*, pages 150–159. IEEE, 2011.
- 17 Konrad K Dabrowski, Petr A Golovach, Pim van’t Hof, Daniël Paulusma, and Dimitrios M Thilikos. Editing to a planar graph of given degrees. In *Computer Science—Theory and Applications*, pages 143–156. Springer, 2015.
- 18 Erik D. Demaine, Timothy D. Goodrich, Kyle Kloster, Brian Lavalley, Quanquan C. Liu, Blair D. Sullivan, Ali Vakilian, and Andrew van der Poel. Structural Rounding: Approximation Algorithms for Graphs Near an Algorithmically Tractable Class. *CoRR*, abs/1806.02771, 2018. [arXiv:1806.02771](https://arxiv.org/abs/1806.02771).
- 19 Rodney G Downey and Michael R Fellows. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013.
- 20 Pål Grønås Drange. *Parameterized Graph Modification Algorithms*. PhD thesis, The University of Bergen, 2015.
- 21 Pål Grønås Drange, Markus S. Dregi, and R. B. Sandeep. Compressing Bounded Degree Graphs. In *LATIN*, volume 9644 of *Lecture Notes in Computer Science*, pages 362–375. Springer, 2016.

- 22 Pål Grønås Drange, Markus Sortland Dregi, Daniel Lokshtanov, and Blair D. Sullivan. On the Threshold of Intractability. In *Proceedings of the 23rd Annual European Symposium on Algorithms*, pages 411–423, Patras, Greece, September 2015.
- 23 Tomáš Ebenlendr, Petr Kolman, and Jiri Sgall. An Approximation Algorithm for Bounded Degree Deletion. *Preprint*, unpublished. URL: <http://kam.mff.cuni.cz/~kolman/papers/star.pdf>.
- 24 Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved Approximation Algorithms for Minimum Weight Vertex Separators. *SIAM Journal on Computing*, 38(2):629–657, 2008.
- 25 Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michal Pilipczuk, and Yngve Villanger. Tight bounds for parameterized complexity of Cluster Editing with a small number of clusters. *Journal of Computer and System Sciences*, 80(7):1430–1447, 2014.
- 26 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-deletion: Approximation, kernelization and optimal FPT algorithms. In *Proceedings of the IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 470–479, 2012.
- 27 Toshihiro Fujito. A unified approximation algorithm for node-deletion problems. *Discrete Applied Mathematics*, 86(2-3):213–231, 1998.
- 28 Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and Applications*, 13(1):113–129, 2010.
- 29 Michael T Gastner and Mark EJ Newman. The spatial structure of networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 49(2):247–252, 2006.
- 30 Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *International Workshop on Parameterized and Exact Computation*, pages 162–173. Springer, 2004.
- 31 Anupam Gupta, Euiwoong Lee, Jason Li, Pasin Manurangsi, and Michał Włodarczyk. Losing Treewidth by Separating Subsets. *CoRR*, abs/1804.01366, 2018. [arXiv:1804.01366](https://arxiv.org/abs/1804.01366).
- 32 Venkatesan Guruswami and Euiwoong Lee. Inapproximability of H-transversal/packing. *SIAM Journal on Discrete Mathematics*, 31(3):1552–1571, 2017.
- 33 Falk Hüffner, Christian Komusiewicz, and André Nichterlein. Editing Graphs Into Few Cliques: Complexity, Approximation, and Kernelization Schemes. In *Proceedings of the 14th International Symposium on Algorithms and Data Structures*, pages 410–421. Springer, 2015.
- 34 Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1802–1811, 2014.
- 35 Ken-ichi Kawarabayashi. Planarity allowing few error vertices in linear time. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 639–648. IEEE, 2009.
- 36 Ken-ichi Kawarabayashi and Anastasios Sidiropoulos. Polylogarithmic Approximation for Minimum Planarization (Almost). In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science*, pages 779–788, Berkeley, CA, October 2017.
- 37 Christian Komusiewicz. Tight Running Time Lower Bounds for Vertex Deletion Problems. *TOCT*, 10(2):6:1–6:18, 2018.
- 38 Michal Kotrbčik, Rastislav Královič, and Sebastian Ordyniak. Edge-Editing to a Dense and a Sparse Graph Class. In *Proceedings of the 12th Latin American Symposium on Theoretical Informatics*, pages 562–575. Springer, 2016.
- 39 Mukkai S Krishnamoorthy and Narsingh Deo. Node-deletion NP-complete problems. *SIAM Journal on Computing*, 8(4):619–625, 1979.
- 40 Euiwoong Lee. Partitioning a graph into small pieces with applications to path transversal. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1546–1558. Society for Industrial and Applied Mathematics, 2017.
- 41 John M. Lewis. On the complexity of the maximum subgraph problem. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 265–274. ACM, 1978.

- 42 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.
- 43 Carsten Lund and Mihalis Yannakakis. The approximation of maximum subgraph problems. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*, pages 40–51. Springer, 1993.
- 44 Avner Magen and Mohammad Moharrami. Robust algorithms for on minor-free graphs based on the Sherali-Adams hierarchy. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 258–271. Springer, 2009.
- 45 Dániel Marx. Parameterized coloring problems on chordal graphs. *Theoretical Computer Science*, 351(3):407–424, 2006.
- 46 Dániel Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.
- 47 Dániel Marx and Ildikó Schlotter. Obtaining a planar graph by vertex deletion. *Algorithmica*, 62(3-4):807–822, 2012.
- 48 Luke Mathieson. The parameterized complexity of editing graphs for bounded degeneracy. *Theoretical Computer Science*, 411(34):3181–3187, 2010.
- 49 Luke Mathieson and Stefan Szeider. Parameterized graph editing with chosen vertex degrees. In *Combinatorial Optimization and Applications*, pages 13–22. Springer, 2008.
- 50 David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30(3):417–427, 1983.
- 51 J. Nešetřil and P. O. de Mendez. *Sparsity: Graphs, Structures, and Algorithms*. Algorithms and Combinatorics. Springer Berlin Heidelberg, 2012.
- 52 Michael Okun and Amnon Barak. A new approach for approximating node deletion problems. *Information Processing Letters*, 88(5):231–236, 2003.
- 53 Bruce A. Reed. Finding Approximate Separators and Computing Tree Width Quickly. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 221–228, 1992.
- 54 Marcus Schaefer. The graph crossing number and its variants: A survey. *The Electronic Journal of Combinatorics*, 1000:21–22, 2013.
- 55 Mingyu Xiao. A Parameterized Algorithm for Bounded-Degree Vertex Deletion. In *Proceedings of the 22nd International Conference on Computing and Combinatorics*, pages 79–91, 2016.
- 56 Mihalis Yannakakis. Node-and edge-deletion NP-complete problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 253–264, 1978.

Dense Peelable Random Uniform Hypergraphs

Martin Dietzfelbinger 

Technische Universität Ilmenau, Germany
martin.dietzfelbinger@tu-ilmenau.de

Stefan Walzer 

Technische Universität Ilmenau, Germany
stefan.walzer@tu-ilmenau.de

Abstract

We describe a new family of k -uniform hypergraphs with independent random edges. The hypergraphs have a high probability of being *peelable*, i.e. to admit no sub-hypergraph of minimum degree 2, even when the edge density (number of edges over vertices) is close to 1.

In our construction, the vertex set is partitioned into linearly arranged *segments* and each edge is incident to random vertices of k consecutive segments. Quite surprisingly, the linear geometry allows our graphs to be peeled “from the outside in”. The density thresholds f_k for peelability of our hypergraphs ($f_3 \approx 0.918$, $f_4 \approx 0.977$, $f_5 \approx 0.992$, ...) are well beyond the corresponding thresholds ($c_3 \approx 0.818$, $c_4 \approx 0.772$, $c_5 \approx 0.702$, ...) of standard k -uniform random hypergraphs.

To get a grip on f_k , we analyse an idealised peeling process on the random weak limit of our hypergraph family. The process can be described in terms of an operator on $[0, 1]^{\mathbb{Z}}$ and f_k can be linked to thresholds relating to the operator. These thresholds are then tractable with numerical methods.

Random hypergraphs underlie the construction of various data structures based on hashing, for instance invertible Bloom filters, perfect hash functions, retrieval data structures, error correcting codes and cuckoo hash tables, where inputs are mapped to edges using hash functions. Frequently, the data structures rely on peelability of the hypergraph, or peelability allows for simple linear time algorithms. Memory efficiency is closely tied to edge density while worst and average case query times are tied to maximum and average edge size.

To demonstrate the usefulness of our construction, we used our 3-uniform hypergraphs as a drop-in replacement for the standard 3-uniform hypergraphs in a retrieval data structure by Botelho et al. [8]. This reduces memory usage from $1.23m$ bits to $1.12m$ bits (m being the input size) with almost no change in running time. Using $k > 3$ attains, at small sacrifices in running time, further improvements to memory usage.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Random Hypergraphs, Peeling Threshold, 2-Core, Hashing, Retrieval, Succinct Data Structure

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.38

1 Introduction

The *core* of a hypergraph $H = (V, E)$ is the largest sub-hypergraph of H with minimum degree at least 2. The core can be obtained by *peeling*, which means repeatedly choosing a vertex of degree 0 or 1 and removing it (and the incident edge if present) from the hypergraph, until no such vertex exists. If the core of H is empty, then H is called *peelable*.

The significance of peelability. Hypergraphs underlie many hashing based data structures and peelability is often necessary for proper operation or allows for simple linear time algorithms. We list a few examples.



© Martin Dietzfelbinger and Stefan Walzer;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 38; pp. 38:1–38:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- **Invertible Bloom Lookup Tables.** IBLTs [22] are based on Bloomier filters [10] which are based on Bloom filters [4]. Each element is inserted in several random positions in a hash table. Any cell stores the XOR of all elements that have been inserted into it. A LIST-ENTRIES query on an IBLT can recover all elements of the table precisely if the underlying hypergraph is peelable. Among other things, IBLTs have been used to construct error correcting codes [34] and to solve the set reconciliation and straggler identification problems [16].
- **Erasur Correcting Codes.** To construct capacity achieving erasure codes, the authors of [28] consider a hypergraph where V corresponds to parity check bits and E to message bits that were lost during transmission. A message bit is incident to precisely those check bits to which it contributed. Correct decoding hinges on peelability of the hypergraph.
- **Cuckoo Hashing and XORSAT.** In the context of cuckoo hash tables [14, 31, 36] and solving random XORSAT formulas [15, 19, 37], (partial) peelability of the underlying hypergraph makes placing all (some) keys or solving the linear system (eliminating some variables) particularly simple.
- **Retrieval and Perfect Hashing.** The retrieval problem (considered later in Section 7) occurs in the context of constructing perfect hash functions [3, 6, 7, 8, 30]. The known approaches involve finding a solution $z : V \rightarrow R$ for a system $(\sum_{v \in e} z(v) = f(e))_{e \in E}$ of equations where $H = (V, E)$ is a hypergraph, $f : E \rightarrow R$ a function and R a small set. If R is a field, then the incidence matrix of H needs to have full rank over R to guarantee the existence of a solution. If H is peelable however, then the existence of a solution is guaranteed even if R only has a group structure. Moreover, it can be computed in linear time.

In these contexts, the hypergraph typically has vertex set $[n] = \{1, \dots, n\}$ and for each element x of an input set S , an edge $e_x \subset [n]$ is created with incidences chosen via hash functions. For theoretical considerations, the edges $(e_x)_{x \in S}$ are often assumed to be independent random variables. This has proven to be a good model for practical settings, even though perfect independence is not achieved by most practical hash functions. An important choice left to the algorithm designer is the distribution of e_x .

Previous work. If the distribution is such that $\mathcal{O}(n)$ edges have size 2 or less (in particular if H is a graph with $\mathcal{O}(n)$ edges), then – due to the well-known “birthday paradox” – there is a constant probability that an edge is repeated. In that case, H is clearly not peelable. The simplest workable candidate for the distribution of e_x is therefore to pick a constant $k \geq 3$ and let e_x contain k vertices chosen independently and uniformly at random. We refer to these standard hypergraphs as *k-uniform Erdős-Renyi hypergraphs* $H_{n,cn}^k$ where c is the *edge density*, i.e. the number of edges over the number of vertices. Corresponding *peelability thresholds* c_k have been determined in [35] meaning if $c < c_k$ then $H_{n,cn}^k$ is peelable with high probability (whp), i.e. with probability approaching 1 as $n \rightarrow \infty$ and if $c > c_k$ then $H_{n,cn}^k$ is not peelable whp. The largest threshold is $c_3 \approx 0.818$. Since the edge density is often tightly linked to a performance metric (e.g. memory efficiency of a dictionary, rate of a code) a density closer to 1 would be desirable, but we know of only two alternative constructions.

To obtain erasure codes with high rates the authors of [28] construct for any $D \in \mathbb{N}$ hypergraphs with edge sizes in $\{5, \dots, D + 4\}$, average edge size $\approx \ln D + 3$ and edge density $1 - 1/D$ that are peelable whp. In particular, this yields peelable hypergraphs with edge densities arbitrarily close to 1. A downside is that the high maximum edge size can lead to worst case query times of $\Theta(D)$ in certain contexts. Motivated by this, the author of

■ **Table 1** The erosion thresholds er_k and peelability thresholds f_k for k -ary fuse graphs satisfy $b_k \leq er_k \leq f_k \leq c_k^*$. The values B_k play a role in Section 5.

k	3	4	5	6	7
b_k	0.9179352469	0.9767692112	0.9924345766	0.9973757381	0.9990561294
c_k^*	0.9179352767	0.9767701649	0.9924383913	0.9973795528	0.9990637588
B_k	0.9179353065	0.9767711186	0.9924422067	0.9973833675	0.9990713882
$\Rightarrow f_k \approx$	0.917935	0.97677	0.99243	0.99738	0.99906

[39] looked into non-uniform hypergraphs with constant maximum edge size. Focusing on hypergraphs with two admissible edge sizes, he found for example that mixing edges of size 3 and size 21 yields a family of hypergraphs with peelability threshold ≈ 0.92 .

Our construction. In this paper we introduce and analyse a new distribution on edges that yields k -uniform hypergraphs with high peelability thresholds that perform well in practical algorithms.

We call our hypergraphs *fuse graphs* (as in the cord attached to a firecracker). There is an underlying linear geometry and similar to how fire proceeds linearly through a lit fuse, the peeling process proceeds linearly through our hypergraphs, in the sense that vertices on the inside of the line tend to only become peelable after vertices closer to the end of the line have already been removed.

Formally, for $k \geq 3$, $\ell \in \mathbb{N}$ and $c \in \mathbb{R}^+$ we define the family $(F(n, k, c, \ell))_{n \in \mathbb{N}}$ of k -uniform fuse graphs as follows. The vertex set is $V = \{1, \dots, n(\ell + k - 1)\}$ where for $i \in I := \{0, \dots, \ell + k - 2\}$ the vertices $\{in + 1, \dots, (i + 1)n\}$ form the i -th *segment*¹. The edge set E has size $cn\ell$. Each edge $e \in E$ is independently determined by one uniformly random variable $j \in J := \{0, \dots, \ell - 1\}$ denoting the *type* of e and k independent random variables o_0, \dots, o_{k-1} uniformly distributed in $[n]$, yielding $e = \{(j + t)n + o_t \mid t \in \{0, \dots, k - 1\}\}$. In other words, e contains one uniformly random vertex from each segment $j, j + 1, \dots, j + k - 1$. There may be repeating edges but the probability that this happens is $\mathcal{O}(1/n)$. The edge density $c \frac{\ell}{\ell + k - 1}$ approaches c for $\ell \gg k$.

Results. Let the *peelability threshold* for k -ary fuse graphs be defined as

$$f_k := \sup\{c \in \mathbb{R}^+ \mid \forall \ell \in \mathbb{N} : \Pr[F(n, k, c, \ell) \text{ is peelable}] \xrightarrow{n \rightarrow \infty} 1\}.$$

Our Main Theorem relates f_k to the *orientability threshold* c_k^* of k -ary Erdős-Rényi hypergraphs and the *erosion threshold* er_k defined in the technical part of our paper.

► **Theorem 1.** *For any $k \geq 3$ we have $er_k \leq f_k \leq c_k^*$.*

The orientability thresholds c_k^* are known exactly [11, 19, 20] and we determine lower bounds on the erosion thresholds er_k . As shown in Table 1, this makes it possible to narrow down f_k to an interval of size 10^{-5} for all $k \in \{3, \dots, 7\}$.

¹ Denoting the segment size by n instead of the number of vertices is more convenient. Note that $|V| = \Theta(n)$ still holds.

Outline. The paper is organised as follows. In Section 2 we idealise the peeling process by switching to the *random weak limit* of our hypergraphs, and capture the essential behaviour of the process in terms of an operator $\hat{\mathbf{P}}$ acting on functions $q : \mathbb{Z} \rightarrow [0, 1]$. For this operator, we identify the properties of being *eroding* and *consolidating* as well as corresponding thresholds er_k and co_k in Section 3. We then prove the “ $\text{er}_k \leq f_k$ ” part of our theorem in Section 4 and give numerical approximations of er_k and co_k in Section 5. The comparatively simple “ $f_k \leq c_k^*$ ” part of our theorem is independent of these considerations and is proved in Section 6. Finally, in Section 7 we demonstrate how using our hypergraphs can improve the performance of practical retrieval data structures.

2 The Peeling Process and Idealised Peeling Operators

In this section we consider how the probabilities for vertices to “survive” $r \in \mathbb{N}$ rounds of peeling changes from one round to the next. In the classical setting this could be described by a function, mapping the old probability to the new one [35]. In our case, however, there are distinct probabilities for each segment of the graph. Thus we need a corresponding operator $\hat{\mathbf{P}}$ that acts on *sequences* of probabilities. Conveniently, it will be independent of n and ℓ .

We almost always suppress n, k, c, ℓ in notation outside of definitions, assuming n to be large. Big- \mathcal{O} notation refers to $n \rightarrow \infty$ while k, c, ℓ are constant.

Consider the parallel peeling process $\text{peel}(F)$ on $F = F(n, k, c, \ell)$. In each *round* of $\text{peel}(F)$, all vertices of degree 0 or 1 are determined and then deleted simultaneously. Deleting a vertex implicitly deletes incident edges. We also define the *rooted peeling process* $\text{peel}_v(F)$ for any vertex $v \in V$, which behaves exactly like $\text{peel}(F)$ except that the special vertex v may only be deleted if it has degree 0, not if it has degree 1. For any $i \in I$ and $r \in \mathbb{N}_0$ we let $q^{(r)}(i) = q^{(r)}(i, n, k, c, \ell)$ be the probability that a vertex v of segment i survives r rounds of $\text{peel}_v(F)$, i.e. is not deleted. Note that the probability is well-defined as vertices of the same segment are symmetric.

By definition, $q^{(0)}(i) = 1$ for all $i \in I$. Whether a vertex v of segment $i \in I$ survives $r > 0$ rounds is a function of its r -neighbourhood $N(n, v, r)$, i.e. the set of vertices and edges of F that can be reached from v by traversing at most r hyperedges.

It is standard to consider the *random weak limit* of F to get a grip on the distribution of $N(n, v, r)$ and thus on $q^{(r)}(i)$. Intuitively, we identify a (possibly infinite) random tree that captures the local characteristics of F for $n \rightarrow \infty$. See [1] for a good survey with examples and details on how to formally define the underlying topology and metric space. In the limit, the binomially distributed vertex degrees (e.g. $\text{Bin}(cn\ell, \frac{1}{n\ell})$ for vertices of segment 0) become Poisson distributed ($\text{Po}(c)$ for segment 0). Short cycles are not only rare but non-existent and certain weakly correlated random variables become perfectly independent.

► **Definition 2 (Limiting Tree).** *Let $k, \ell \in \mathbb{N}$, $c \in \mathbb{R}^+$ and $i \in I$. The random (possibly infinite) hypertree $T_i = T_i(k, c, \ell)$ is distributed as follows.*

T_i has a root vertex $\text{root}(T_i)$ of segment² i which for each $j \in \{i - k + 1, \dots, i\} \cap J$ has $d_j \sim \text{Po}(c)$ child edges of type j . Each child edge of type j is incident to $k - 1$ (fresh) child vertices of its own, one for each segment $i' \in \{j, \dots, j + k - 1\} \setminus \{i\}$. The sub-hypertree at such a child vertex of segment i' is distributed recursively (and independently of its sibling-subtrees) according to $T_{i'}$.

² In the current context, the segment of a vertex is an abstract label. There can be an unbounded number of vertices of each segment.

Since all arguments are standard in contexts where local weak convergence plays a role, we state the following lemma without proof. For instance, a full argument to show a similar convergence is given in [25]. See also [24] for the related technique of Poissonisation.

► **Lemma 3.** *Let $r \in \mathbb{N}$ be constant. Let further $N(n, v, r)$ be the r -neighbourhood of a vertex v of segment i in F and $T_i^{(r)}$ the r -neighbourhood of $\text{root}(T_i)$, both viewed as undirected and unlabelled hypergraphs. Then $N(n, v, r)$ converges in distribution to $T_i^{(r)}$ as $n \rightarrow \infty$.*

We now direct our attention to survival probabilities in the idealised peeling processes $(\text{peel}_{\text{root}(T_i)}(T_i))_{i \in I}$, which are easier to analyse than those of $\text{peel}_v(F)$.

► **Lemma 4.** *Let $r \in \mathbb{N}_0$ be constant and $q_T^{(r)}(i) = q_T^{(r)}(i, k, c, \ell)$ be the probability that $\text{root}(T_i)$ survives r rounds of $\text{peel}_{\text{root}(T_i)}(T_i)$ for $i \in I$. Then*

$$q_T^{(r+1)}(i) = 1 - \exp\left(-c \sum_{j \in \{i-k+1, \dots, i\} \cap J} \prod_{\substack{j \leq i' < j+k \\ i' \neq i}} q_T^{(r)}(i')\right) \quad \text{for } i \in I.$$

Proof. Let $i \in I$ and $v = \text{root}(T_i)$. Assume $j \in \{i-k+1, \dots, i\} \cap J$ is the type of some edge e incident to v . Edge e survives r rounds of $\text{peel}_v(T_i)$ if and only if all of its incident vertices survive these r rounds. Since v itself may not be deleted by $\text{peel}_v(T_i)$ as long as e exists, the relevant vertices are the $k-1$ child vertices, one for each segment $i' \in \{j, \dots, j+k-1\} - \{i\}$. Call these w_1, \dots, w_{k-1} and denote the subtrees rooted at those vertices by W_1, \dots, W_{k-1} . Now consider the peeling processes $\text{peel}_{w_1}(W_1), \dots, \text{peel}_{w_{k-1}}(W_{k-1})$. Assume one of them, say $\text{peel}_{w_s}(W_s)$, deletes w_s in round $r' \leq r$, meaning w_s has degree 0 before round r' . It follows that w_s has degree at most 1 before round r' in $\text{peel}_v(T_i)$, meaning $\text{peel}_v(T_i)$ deletes e in round r' (or earlier). Conversely, if none of $\text{peel}_{w_1}(W_1), \dots, \text{peel}_{w_{k-1}}(W_{k-1})$ delete their root vertex within r rounds, then w_1, \dots, w_{k-1} have degree at least 2 after round r of $\text{peel}_v(T_i)$ and e survives round r of $\text{peel}_v(T_i)$. This makes the probability for e to survive r rounds of $\text{peel}_v(T_i)$ equal to $p_{ij} := \prod_{j \leq i' < j+k, i' \neq i} q_T^{(r)}(i')$. Since the number m_{ij} of edges of type j incident to v has distribution $m_{ij} \sim \text{Po}(c)$, the number m'_{ij} of edges of type j incident to v surviving r rounds of $\text{peel}_v(T_i)$ is a correspondingly *thinned out* variable, namely $m'_{ij} \sim \text{Bin}(m_{ij}, p_{ij})$, which means $m'_{ij} \sim \text{Po}(cp_{ij})$.

The claim now follows by observing that v survives $r+1$ rounds of $\text{peel}_v(T_i)$ if and only if at least one of its child edges survives r rounds of $\text{peel}_v(T_i)$:

$$\begin{aligned} q_T^{(r+1)}(i) &= \Pr[v \text{ survives } r+1 \text{ rounds of } \text{peel}_v(T_i)] = 1 - \Pr\left[\bigcap_{j \in \{i-k+1, \dots, i\} \cap J} \{m'_{ij} = 0\}\right] \\ &= 1 - \prod_{j \in \{i-k+1, \dots, i\} \cap J} \Pr[m'_{ij} = 0] = 1 - \prod_{j \in \{i-k+1, \dots, i\} \cap J} \exp(-cp_{ij}) = 1 - \exp\left(-c \sum_{j \in \{i-k+1, \dots, i\} \cap J} p_{ij}\right). \end{aligned}$$

Replacing p_{ij} with its definition completes the proof. ◀

For convenience we define, for $k \geq 3, \ell \in \mathbb{N}$ and $c \in \mathbb{R}^+$, the operator $\mathbf{P} = \mathbf{P}(k, c, \ell)$, which maps any $q : I \rightarrow [0, 1]$ to $\mathbf{P}q : I \rightarrow [0, 1]$ with

$$(\mathbf{P}q)(i) = 1 - \exp\left(-c \sum_{j \in \{i-k+1, \dots, i\} \cap J} \prod_{\substack{j \leq i' < j+k \\ i' \neq i}} q(i')\right) \quad \text{for } i \in I.$$

Together Lemmas 3 and 4 imply that \mathbf{P} can be used to approximate survival probabilities.

► **Corollary 5.** *Let $r \in \mathbb{N}_0$ be constant. Then for all $i \in I$*

$$\mathbf{P}^r q^{(0)}(i) \stackrel{\text{def}}{=} \mathbf{P}^r q_T^{(0)}(i) \stackrel{\text{Lem 4}}{=} q_T^{(r)}(i) \stackrel{\text{Lem 3}}{=} q^{(r)}(i) \pm o(1).$$

To obtain *upper* bounds on survival probabilities, we may remove the awkward restriction “ $\cap J$ ” in the definition of \mathbf{P} . We define $\hat{\mathbf{P}} = \hat{\mathbf{P}}(k, c)$ as mapping $q : \mathbb{Z} \rightarrow [0, 1]$ to $\hat{\mathbf{P}}q : \mathbb{Z} \rightarrow [0, 1]$ with

$$(\hat{\mathbf{P}}q)(i) = 1 - \exp\left(-c \sum_{j=i-k+1}^i \prod_{\substack{j \leq i' < j+k \\ i' \neq i}} q(i')\right) \quad \text{for } i \in \mathbb{Z}.$$

Note that $\hat{\mathbf{P}}$ does not depend on ℓ or n . To simplify notation, we assume that the old operator \mathbf{P} also acts on functions $q : \mathbb{Z} \rightarrow [0, 1]$, ignoring $q(i)$ for $i \notin I$, and producing $\mathbf{P}q : \mathbb{Z} \rightarrow [0, 1]$ with $\mathbf{P}q(i) = 0$ for $i \notin I$. We also extend $q^{(0)}$ to be $\mathbf{1}_I : \mathbb{Z} \rightarrow [0, 1]$, i.e. the characteristic function on I , essentially introducing vertices of segments $i \notin I$ which are, however, already deleted with probability 1 before the first round begins. Note that while $q^{(r)}(i)$ and $q_T^{(r)}(i)$ are by definition non-increasing in r , this is not the case for $(\hat{\mathbf{P}}^r q^{(0)})(i)$. For instance, $\hat{\mathbf{P}}^r q^{(0)}$ has support $\{-r, -r+1, \dots, \ell+k-2+r\}$, which grows with r .³ The following lemma lists a few easily verified properties of $\hat{\mathbf{P}}$. All inequalities between functions should be interpreted point-wise.

► **Lemma 6.**

- (i) $\forall q : \mathbb{Z} \rightarrow [0, 1] : \mathbf{P}q \leq \hat{\mathbf{P}}q$.
- (ii) $\hat{\mathbf{P}}$ commutes with the shift operators \ll and \gg defined via $(\ll q)(i) = q(i+1)$ and $(\gg q)(i) = q(i-1)$. In other words, we have $\forall q : \mathbb{Z} \rightarrow [0, 1] : \hat{\mathbf{P}}(\ll q) = \ll(\hat{\mathbf{P}}q) \wedge \hat{\mathbf{P}}(\gg q) = \gg(\hat{\mathbf{P}}q)$.
- (iii) $\hat{\mathbf{P}}$ is monotonic, i.e. $\forall q, q' : \mathbb{Z} \rightarrow [0, 1] : q \leq q' \Rightarrow \hat{\mathbf{P}}q \leq \hat{\mathbf{P}}q'$.
- (iv) $\hat{\mathbf{P}}$ respects monotonicity, i.e. if $q(i)$ is (strictly) increasing in i , then so is $(\hat{\mathbf{P}}q)(i)$.

3 Two Fixed Points Battling for Territory

In this section we define the *erosion* and *consolidation thresholds* at which the behaviour of $\hat{\mathbf{P}}$ changes in crucial ways.

First, we require a few facts about the function $f : [0, 1] \rightarrow [0, 1]$ mapping $x \mapsto 1 - e^{-ckx^{k-1}}$. It appears in the analysis of cores in k -ary Erdős-Renyi hypergraphs $H_{n, cn}^k$, essentially mapping the probability ρ_r for a vertex to survive r rounds of peeling to the probability $\rho_{r+1} = f(\rho_r)$ to survive $r+1$ rounds of peeling, see [35, page 5]⁴.

The threshold c_k for the appearance of a core in $H_{n, cn}^k$ turns out to be the threshold for the appearance of a non-zero fixed point of f . The following is implicit in the analysis.

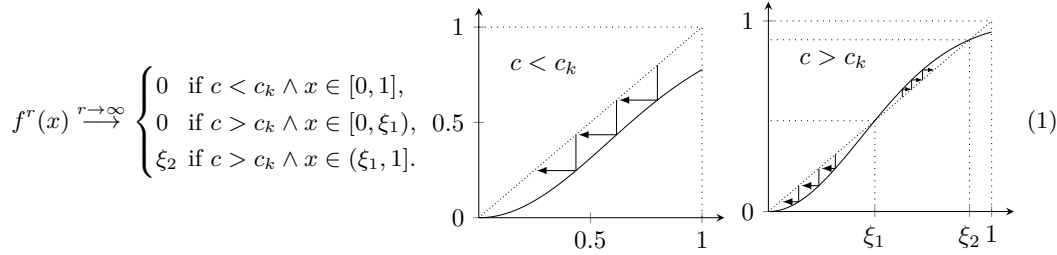
► **Fact 7** ([35, Proofs of Lemmas 3 and 4]).

- (i) For $c < c_k$, f has only the fixed point $f(0) = 0$, with $f'(0) < 1$.
- (ii) For $c > c_k$, there are exactly three fixed points 0 , $\xi_1 = \xi_1(k, c)$ and $\xi_2 = \xi_2(k, c)$ where $f'(\xi_1) > 1$ while $f'(0), f'(\xi_2) < 1$.

³ It is still possible to interpret $\hat{\mathbf{P}}^r q^{(0)}(i)$ as survival probabilities in more symmetric extended versions \hat{T}_i of the tree T_i , but we will not pursue this.

⁴ Our setting corresponds to the choices $(r_{\text{Molloy}}, k_{\text{Molloy}}, c_{\text{Molloy}}) = (k, 2, c \cdot (k-1)!)$.

This implies the following behaviour of applying f repeatedly to a starting value x . This should be immediately clear from the sketches on the right.



Note that f captures the behaviour of $\hat{\mathbf{P}}$ on constant functions $\text{const}_x(i) := x$, in the sense that $\hat{\mathbf{P}}\text{const}_x = \text{const}_{f(x)}$. For $c < c_k$ we therefore have for all $i \in I$

$$\mathbf{P}^r q^{(0)}(i) \stackrel{\text{Cor 5}}{=} q^{(r)}(i) \pm o(1) \text{ and } \mathbf{P}^r q^{(0)} \leq \hat{\mathbf{P}}^r q^{(0)} \leq \hat{\mathbf{P}}^r \text{const}_1 = \text{const}_{f^r(1)} \xrightarrow{r \rightarrow \infty} \text{const}_0.$$

In conjunction with a later lemma, this is sufficient to show that F is peelable whp in this case. A similar argument for $c = c_k$ is possible as well. Our focus from now on is therefore on the interesting case $c > c_k$ where the three distinct fixed points $0, \xi_1, \xi_2$ of f exist.

We give an intuitive account of the phenomenon underlying the following steps before continuing formally. Due to (1) we have

$$\hat{\mathbf{P}}^r \text{const}_x \xrightarrow{r \rightarrow \infty} \begin{cases} \text{const}_0 & \text{for } x < \xi_1 \\ \text{const}_{\xi_2} & \text{for } x > \xi_1. \end{cases}$$

Now consider what happens if we iterate $\hat{\mathbf{P}}$ on a function that is “torn” between these two cases. Concretely, let us consider the function step_0^1 where we define $\text{step}_x^y : \mathbb{Z} \rightarrow [0, 1]$ to have value y on \mathbb{N}_0 and value x on negative inputs. Should we expect $\hat{\mathbf{P}}^r \text{step}_0^1$ to converge to const_0 or const_{ξ_2} as r increases? It turns out both is possible, depending on c .

Speaking more generally, let $q : \mathbb{Z} \rightarrow [0, 1]$ be any function. If $N(i) := \{i - k + 1, \dots, i + k - 1\} \setminus \{i\}$ then $\hat{\mathbf{P}}q(i)$ depends (monotonically) on $(q(i'))_{i' \in N(i)}$. It is clear that if $q(i') < \xi_1$ for all $i' \in N(i)$, then $\hat{\mathbf{P}}q(i) < \xi_1$ as well. Similarly, if $q(i') > \xi_1$ for all $i' \in N(i)$ then $\hat{\mathbf{P}}q(i) > \xi_1$. If, however, there are indices $i'_1, i'_2 \in N(i)$ with $q(i'_1) < \xi_1 < q(i'_2)$ then $\hat{\mathbf{P}}q(i)$ could be above or below ξ_1 ; in this case we call the index i *contested*.

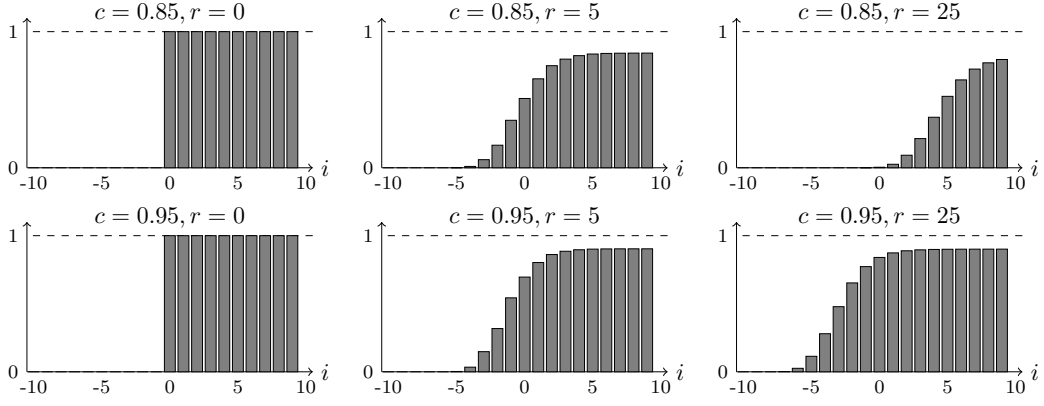
The contested area of step_0^1 is $[-k + 1, k - 2]$. Iterating $\hat{\mathbf{P}}$ we obtain $\hat{\mathbf{P}}^r \text{step}_0^1$ for $r \in \mathbb{N}_0$. For all $r \in \mathbb{N}_0$ the contested area is an interval of size $2k - 2$ with all values to the left of it (towards $-\infty$) less than ξ_1 and all values to the right of it (towards ∞) bigger than ξ_1 . However, the contested area may *shift*. If the domain of values bigger than ξ_1 is shrinking (“*eroding*”), then we see convergence to const_0 . If conversely it is growing (“*consolidating*”), then we see convergence to const_{ξ_2} . In Figure 1 we visualise these effects. There is only a small range of values c where both fixed points seem equally “strong” and the same area remains perpetually contested.

With this in mind, we make the following definitions. For a compact formulation in the coarse terms of shifts (“ \ll ”, “ \gg ”) and point-wise inequalities (“ $<$ ”, “ $>$ ”) we use slightly different step functions.

► **Definition 8.** Let $k \geq 3, c \in \mathbb{R}^+$ and $\hat{\mathbf{P}} = \hat{\mathbf{P}}(k, c)$ as above. We say

$$\hat{\mathbf{P}} \text{ is eroding if } \exists R \in \mathbb{N} : \hat{\mathbf{P}}^R \text{step}_{\xi_1/2}^1 < \gg \text{step}_{\xi_1/2}^1$$

$$\text{and } \hat{\mathbf{P}} \text{ is consolidating if } \exists R \in \mathbb{N} : \hat{\mathbf{P}}^R \text{step}_0^{(\xi_1+\xi_2)/2} > \ll \text{step}_0^{(\xi_1+\xi_2)/2}.$$



■ **Figure 1** Depiction of $\hat{\mathbf{P}}^r \text{step}_0^1$ for $c \in \{0.85, 0.95\}$ and $r \in \{0, 5, 25\}$ on the range $i \in \{-10, \dots, 10\}$. The phenomenon of *erosion* can be seen on the top with the plot seemingly moving towards the right between $r = 5$ and $r = 25$. Similarly, *consolidation* can be seen on the bottom.

We define the corresponding *erosion* and *consolidation thresholds* as

$$\text{er}_k = \sup\{c \in \mathbb{R}^+ \mid \hat{\mathbf{P}}(k, c) \text{ is eroding}\}, \quad \text{co}_k = \inf\{c \in \mathbb{R}^+ \mid \hat{\mathbf{P}}(k, c) \text{ is consolidating}\}.$$

Note that $c < \text{er}_k$ implies that $\hat{\mathbf{P}}(k, c)$ is eroding and $c > \text{co}_k$ implies $\hat{\mathbf{P}}(k, c)$ is consolidating as would be expected. This uses that the definition of $\hat{\mathbf{P}}$ is monotonic in c .

The following lemma states that erosion (consolidation) are sufficient conditions for const_0 (const_{ξ_2}) to “win the battle” when iterating $\hat{\mathbf{P}}$ on step_0^1 .

► **Lemma 9.** *Let $k \geq 3$.*

- (i) *If $c < \text{er}_k$ and $i \in \mathbb{Z}$, then $\hat{\mathbf{P}}^r \text{step}_0^1(i) \xrightarrow{r \rightarrow \infty} 0$.*
- (ii) *If $c > \text{co}_k$ and $i \in \mathbb{Z}$, then $\hat{\mathbf{P}}^r \text{step}_0^1(i) \xrightarrow{r \rightarrow \infty} \xi_2$.*
- (iii) $\text{er}_k \leq \text{co}_k$.

Proof.

- (i) Let $R \in \mathbb{N}$ be the witness to the fact that $\hat{\mathbf{P}}(k, c)$ is eroding and $i \in \mathbb{Z}$ arbitrary.

$$\begin{aligned} \lim_{r \rightarrow \infty} (\hat{\mathbf{P}}^r \text{step}_0^1)(i) &\leq \lim_{r \rightarrow \infty} (\hat{\mathbf{P}}^r \text{step}_{\xi_1/2}^1)(i) = \lim_{r \rightarrow \infty} (\hat{\mathbf{P}}^r ((\hat{\mathbf{P}}^R)^{kr} \text{step}_{\xi_1/2}^1))(i) \\ &\leq \lim_{r \rightarrow \infty} (\hat{\mathbf{P}}^r (\gg^{kr} \text{step}_{\xi_1/2}^1))(i) = \lim_{r \rightarrow \infty} (\gg^{kr} (\hat{\mathbf{P}}^r \text{step}_{\xi_1/2}^1))(i) \\ &= \lim_{r \rightarrow \infty} (\hat{\mathbf{P}}^r \text{step}_{\xi_1/2}^1)(i - kr) = \lim_{r \rightarrow \infty} (\hat{\mathbf{P}}^r \text{const}_{\xi_1/2})(i - kr) \\ &= \lim_{r \rightarrow \infty} \text{const}_{f^r(\xi_1/2)}(i - kr) = \lim_{r \rightarrow \infty} f^r(\xi_1/2) = 0. \end{aligned}$$

When replacing $\text{step}_{\xi_1/2}^1$ by $\text{const}_{\xi_1/2}$ we exploited that $(\hat{\mathbf{P}}^r q)(i)$ depends only on the values $q(i')$ for $i' \in \{i - k + 1, \dots, i + k - 1\}$ and thus $(\hat{\mathbf{P}}^r q)(i)$ depends only on the values $q(i')$ for $i' \in [i - (k - 1)r, i + (k - 1)r]$.

- (ii) The proof is analogous to the proof of (i).
- (iii) This is clear, since the implications of (i) and (ii) are mutually exclusive. ◀

4 Erosion is Sufficient for Peeling

We now connect the phenomenon of erosion to the survival probabilities $q^{(R)}(i)$ we were originally interested in. For $c < \text{er}_k$ and any $\ell \in \mathbb{N}$, they can be made smaller than any $\delta > 0$ in $R = R(\delta, \ell)$ rounds. For $c > \text{co}_k$ and ℓ sufficiently large, no constant number of rounds suffices to reduce all survival probabilities below ξ_1 .

► **Lemma 10.** *Let $k \geq 3$.*

- (i) *If $c < \text{er}_k$ then $\forall \ell \in \mathbb{N}, \delta > 0: \exists R, N \in \mathbb{N}: \forall n \geq N, i \in I: q^{(R)}(i) < \delta$.*
- (ii) *If $c > \text{co}_k$ then $\exists L = L(k, c): \forall \ell \geq L: \exists i \in I: \lim_{r \rightarrow \infty} \lim_{n \rightarrow \infty} q^{(r)}(i) > \xi_1$.*

Proof.

- (i) Let $\ell \in \mathbb{N}$ and $\delta > 0$ be arbitrary constants. Using (i) from Lemma 9, there exists a constant R such that $\hat{\mathbf{P}}^R \text{step}_0^1(i) \leq \delta/2$ for all $i \in I$. Therefore for $i \in I$:

$$q^{(R)}(i) \stackrel{\text{Cor 5}}{=} (\mathbf{P}^R q^{(0)})(i) + o(1) \leq (\hat{\mathbf{P}}^R q^{(0)})(i) + o(1) \leq (\hat{\mathbf{P}}^R \text{step}_0^1)(i) \leq \delta/2 + o(1).$$

which implies the existence of an appropriate $N \in \mathbb{N}$.

- (ii) Let $R \in \mathbb{N}$ be the witness to the fact that $\hat{\mathbf{P}}(k, c)$ is consolidating and let $\ell \geq L(k, c) := 4d$ for $d = (k - 1)R$. Consider the function $q^* : \mathbb{Z} \rightarrow [0, 1]$ defined as $q^* = \mathbb{1}_{\{d, \dots, \ell - d - 1\}} \cdot (\xi_1 + \xi_2)/2$, i.e. the function with value $(\xi_1 + \xi_2)/2$ on its support $\{d, \dots, \ell - d - 1\}$ and 0 outside of it. For any $d \leq i < \ell - 2d$ we have

$$\begin{aligned} \mathbf{P}^R q^*(i) &= \hat{\mathbf{P}}^R q^*(i) = \hat{\mathbf{P}}^R \text{step}_0^{(\xi_1 + \xi_2)/2}(i - d) \geq \llcorner \text{step}_0^{(\xi_1 + \xi_2)/2}(i - d) \\ &= (\xi_1 + \xi_2)/2 = q^*(i). \end{aligned}$$

For the first equality, we exploited that i is so far from the borders of $I = \{0, \dots, \ell - 1\}$ that there is no difference between \mathbf{P} and $\hat{\mathbf{P}}$. For the second equality we used that only the values of q^* on $\{i - d, \dots, i + d\}$ play a role and q^* is a (shifted) step function on that domain. By mirroring, the same argument can be made to get $\mathbf{P}^R q^*(i) \geq q^*(i)$ for $2d \leq i < \ell - d$ as well and thus the point-wise inequality $\mathbf{P}^R q^* \geq q^*$. Since $q^{(0)} \geq q^*$ we get

$$\lim_{r \rightarrow \infty} \lim_{n \rightarrow \infty} q^{(r)} \stackrel{\text{Cor 5}}{=} \lim_{r \rightarrow \infty} \mathbf{P}^r \mathbb{1}_I \geq \lim_{r \rightarrow \infty} \mathbf{P}^r q^* \geq q^*.$$

Since q^* exceeds ξ_1 on $\{d, \dots, \ell - d - 1\}$, this implies the claim. ◀

While Lemma 10(i) is sufficient to show that all but a δ -fraction of the vertices is peeled whp if $c < \text{er}_k$, we still need the following combinatorial argument that shows that whp no non-empty core is contained within the remaining vertices. Arguments such as these are standard, many similar ones can be found for instance in [18, 19, 23, 27, 29, 35, 32].

► **Lemma 11.** *For any $k \geq 3$, $\ell \in \mathbb{N}$ and $c \in (0, 1)$ there exists $\delta = \delta(k, \ell) > 0$ such that the following holds whp. For any non-empty set $V' \subseteq V$ of at most $\delta|V|$ vertices of $F = (V, E)$, there exists $v \in V'$ of degree at most 1 in the sub-hypergraph of H induced by V' .*

Proof. In the course of the proof we will implicitly encounter positive upper bounds on δ in terms of k and ℓ . Any $\delta > 0$ small enough to respect these bounds is suitable. We consider the events $(W_{s,t})_{k \leq s \leq \delta|V|, \frac{2s}{k} \leq t \leq |E|}$ that some small set V' of size s induces t edges. If none of these events occurs, then all such V' induce less than $2|V'|/k$ edges and therefore induce hypergraphs with average degree less than 2, so a vertex of degree at most 1 exists in each of them.

It is thus sufficient to show that $\Pr[\bigcup_s \bigcup_t W_{s,t}] = \mathcal{O}(1/n)$. We shall use a first moment argument. First note that F has duplicate edges with probability $\binom{cn\ell}{2} (\ell n^k)^{-1} = \mathcal{O}(n^{-1})$, so we restrict our attention to F without duplicate edges. Given s and t there are $\binom{(\ell+k-1)n}{s}$ ways to choose V' and at most $\binom{s^k}{t}$ ways to choose which k -tuples of vertices in V' induce an edge. The probability that any given k -tuple actually does induce an edge is either zero if the k vertices are not of consecutive segments or $1 - (1 - (\ell n^k)^{-1})^{cn\ell} \leq \frac{cn}{n^k} = \frac{1}{n^{k-1}}$. Thus, using constants $C, C', C'', C''' \in \mathbb{R}^+$ (that may depend on k and ℓ) where precise values do not matter, we get

$$\begin{aligned}
 \Pr\left[\bigcup_{s=k}^{\delta|V|} \bigcup_{t=\frac{2s}{k}}^{|E|} W_{s,t}\right] &\leq \sum_{s=k}^{\delta|V|} \sum_{t=\frac{2s}{k}}^{|E|} \Pr[W_{s,t}] \leq \sum_{s=k}^{\delta|V|} \sum_{t=\frac{2s}{k}}^{|E|} \binom{(\ell+k-1)n}{s} \binom{s^k}{t} \left(\frac{1}{n^{k-1}}\right)^t \\
 &\leq \sum_{s=k}^{\delta|V|} \sum_{t=\frac{2s}{k}}^{|E|} \left(\frac{e(\ell+k-1)n}{s}\right)^s \left(\frac{es^k}{tn^{k-1}}\right)^t \leq \sum_{s=k}^{\delta|V|} \sum_{t=\frac{2s}{k}}^{|E|} \left(C\frac{n}{s}\right)^s \left(C'\frac{s^{k-1}}{n^{k-1}}\right)^t \\
 &\leq 2 \sum_{s=k}^{\delta|V|} \left(C\frac{n}{s}\right)^s \left(C'\frac{s^{k-1}}{n^{k-1}}\right)^{\frac{2s}{k}} = 2 \sum_{s=k}^{\delta|V|} \left(C''\frac{n^k s^{2k-2}}{s^k n^{2k-2}}\right)^{\frac{s}{k}} = 2 \sum_{s=k}^{\delta|V|} \left(C'''\frac{s}{n}\right)^{\frac{s(k-2)}{k}}.
 \end{aligned}$$

To get rid of the summation over t , we assumed $(s/n)^{k-1} \leq \delta^{k-1} \leq \frac{1}{2C'}$. Elementary arguments show that in the resulting bound, the contribution of summands for $s \in \{k, \dots, 2k\}$ is of order $\mathcal{O}(\frac{1}{n})$, the contribution of the summands with $s \in \{2k+1, \dots, \mathcal{O}(\log n)\}$ are of order $\mathcal{O}(\frac{\log n}{n^2})$ (using $\frac{s}{n} \leq \frac{\log n}{n}$) and the contribution of the remaining terms with $s \geq 3 \log_2 n$ is of order $\mathcal{O}(2^{-\log_2 n}) = \mathcal{O}(\frac{1}{n})$ (using $C'''\frac{s}{n} \leq C'''\delta(\ell+2) \leq \frac{1}{2}$).

This gives $\Pr[\bigcup_{s,t} W_{s,t}] = \mathcal{O}(n^{-1})$, proving the claim. \blacktriangleleft

We are ready to prove the “ $\text{er}_k \leq f_k$ ” of Theorem 1, stated here as a theorem of its own.

► **Theorem 12.** *For all $k \geq 3$ we have $\text{er}_k \leq f_k$.*

Proof. We need to prove that for any $c < \text{er}_k$ and any $\ell \in \mathbb{N}$ the fuse graph $F = F(n, k, c, \ell)$ is peelable whp.

First, let $\delta = \delta(k, \ell)$ be the constant from Lemma 11 and $R = R(\delta/2, \ell)$ as well as $N = N(\delta/2, \ell)$ the corresponding constants from Lemma 10(i).

Assuming $n \geq N$ we have $q^{(R)}(i) \leq \delta/2$ for all $i \in I$, meaning any vertex v from F is *not* deleted within R rounds of $\text{peel}_v(F)$ with probability at most $\delta/2$. Since $\text{peel}(F)$ deletes at least the vertices that any $\text{peel}_v(F)$ for $v \in V$ deletes, the expected number of vertices not deleted by $\text{peel}(F)$ within R rounds is at most $\delta|V|/2$.

Now standard arguments using Azuma’s inequality (see e.g. [33, Theorem 13.7]) suffice to conclude that whp at most $\delta|V|$ vertices are not deleted by $\text{peel}(F)$ within R rounds.

By Lemma 11 whp neither the remaining $\delta|V|$ vertices, nor any of its subsets induces a hypergraph of minimum degree 2. Therefore the core of F is empty. \blacktriangleleft

A natural follow-up question to Theorem 12 would be whether $\text{er}_k = f_k$, which would also imply $f_k \leq \text{co}_k$. To establish this stronger claim, we would have to exclude the possibility that for certain densities c there is a function $r(n) = \omega(1)$ such that a constant fraction of vertices survive $r(n)$ rounds but are nevertheless deleted eventually. It seems plausible that arguments similar to [35, Lemma 4] can be used, but since our main goal is reached we do not pursue this now.

5 Approximating the Erosion and Consolidation Thresholds

We now approximate the thresholds er_k (and analogously co_k) with numerical methods. Note that if $c < \text{er}_k$ (if $c > \text{co}_k$), then this can be verified in a finite computation, because the correct value of R , together with a bound on the required precision of floating point operations (when rounding conservatively), constitutes a witness. Moreover, the function $\hat{\mathbf{P}}^r \text{step}_{\xi, 1/2}^1$ can be represented by a finite number of reals, since it is constant on $(-\infty, -(k-1)r]$ and constant on $[(k-1)r, \infty)$.

To approximate er_k (and co_k) with high precision, more efficient approaches are required, however. We compute upper bounds on $\hat{\mathbf{P}}^r \text{step}_{\xi_1/2}^1$ by focusing on a finite domain $[-D, D]$ for some $D \in \mathbb{N}$ and rounding conservatively outside of it. Concretely we define $(a_r : \mathbb{Z} \rightarrow [0, 1])_{r \in \mathbb{N}_0}$ (dependent on k, c and D) with $a_0 := \text{step}_{\xi_1/2}^1$ (analogously $(b_r : \mathbb{Z} \rightarrow [0, 1])_{r \in \mathbb{N}_0}$ with $b_0 := \text{step}_0^{(\xi_1 + \xi_2)/2}$). For $r \geq 0$ we let

$$a_{r+1}(i) := \begin{cases} a_{r+1}(-D) & \text{if } i < -D, \\ \hat{\mathbf{P}}a_r(i) & \text{if } -D \leq i \leq D, \\ 1 & \text{if } i > D. \end{cases} \quad b_{r+1}(i) := \begin{cases} 0 & \text{if } i < -D, \\ \hat{\mathbf{P}}b_r(i) & \text{if } -D \leq i \leq D, \\ \hat{\mathbf{P}}b_r(D) & \text{if } i > D. \end{cases}$$

Due to the limited effective domain, each a_r is given by $2D + 2$ values. It is easy to see that each a_r is monotonous and fulfils $a_{r+1} \leq \hat{\mathbf{P}}a_r$, which implies $\hat{\mathbf{P}}^r \text{step}_{\xi_1/2}^1 \leq a_r$. If we find $a_r(0) < \xi_1/2$, then by monotonicity we have $a_r \leq \gg \text{step}_{\xi_1/2}^1$ and therefore:

$$\exists R \in \mathbb{N} : a_R(0) < \xi_1/2 \quad \Rightarrow \quad \exists R \in \mathbb{N} : \hat{\mathbf{P}}^R \text{step}_{\xi_1/2}^1 < \gg \text{step}_{\xi_1/2}^1 \stackrel{\text{def}}{\Rightarrow} c < \text{er}_k.$$

(Analogously if $b_R(-1) > (\xi_1 + \xi_2)/2$ then $c > \text{co}_k$ follows.)

Experimental Results. For $D = 50$ and all $k \in \{3, \dots, 7\}$ we computed, using double-precision floating point values, a_1, a_2, \dots and b_1, b_2, \dots for various c . For each pair (k, c) , we either find that $\hat{\mathbf{P}}(k, c)$ is consolidating, it is eroding, or none of the two can be verified. The results suggest that $\text{er}_k < c_k^* < \text{co}_k$ where c_k^* is the orientability threshold for k -ary Erdős-Renyi hypergraphs.

Concretely, we considered for $j = 1, 2, 3, \dots$ the values $c_k^* - 2^{-j}$ and tried to verify that they are less than er_k . The largest for which we succeeded is reported as b_k in Table 1 on page 3. The largest number of iterations required was $6 \cdot 10^7$. For the first value that could not be shown to be less than er_k , our approximations of the sequence of $(a_i)_{i \in \mathbb{N}}$ became stationary with $a[0] > \xi_1/2$, i.e. the double-precision floats did not change any more (the highest number of iterations to reach this point was $2 \cdot 10^8$). It is possible that the value is still less than er_k and our choice of D or the precision of our floats is simply insufficient. Further experiments with 128-bit floats and larger values of D suggest however, that there is a tiny but real gap between er_k, c_k^* and co_k and the natural conjecture of equality is misplaced.

In the same way we report the smallest value of the form $c_k^* + 2^{-j}$ for which we verified that it exceeds co_k as B_k in Table 1.

6 Peeling Necessitates Orientability of Erdős-Renyi Hypergraphs

We now prove the “ $f_k \leq c_k^*$ ”-half of Theorem 1, stated as Theorem 14. Recall that an *orientation* of a hypergraph $H = (V, E)$ is an injective map $f : E \rightarrow V$ with $f(e) \in e$ for all $e \in E$ and that c_k^* is the threshold for orientability of k -uniform Erdős-Renyi hypergraphs.

After classical (2-ary) cuckoo hashing was discovered [36] (relying on $c_2^* = \frac{1}{2}$), the thresholds for $k > 2$ were determined independently by [11, 19, 20], with generalisations to other graphs and hypergraphs studied in [9, 17, 25, 26, 40].

Note that if H is peelable then it is also orientable: Just orient each edge e to a vertex $v \in e$ such that v and e are deleted in the same round of $\text{peel}(H)$.

Our proof of Theorem 14 relies strongly on a deep and remarkable theorem due to Lelarge [27]. To clarify its role in our proof, we restate it in weaker but sufficient form.

38:12 Dense Peelable Hypergraphs

► **Theorem 13** (Lelarge [27, Theorem 4.1]). *Let $(G_n = (A_n, B_n, E_n))_{n \in \mathbb{N}}$ be a sequence of bipartite graphs with $|E_n| = O(|A_n|)$. Let further $M(G_n)$ be the size of a maximum matching in G_n . If the random weak limit ρ of $(G_n)_{n \in \mathbb{N}}$ is a bipartite unimodular Galton-Watson tree, then $\lim_{n \rightarrow \infty} \frac{M(G_n)}{|A_n|}$ exists almost surely and depends only on ρ .*

To see the connection, note that an orientation of a hypergraph is a left-perfect matching in its (bipartite) incidence graph.

► **Theorem 14.** *For all $k \geq 3$ we have $f_k \leq c_k^*$.*

Proof. Let $c = c_k^* + \varepsilon$. We need to show that there exists $\ell \in \mathbb{N}$ such that the fuse graph $F = F(n, k, c, \ell)$ is not peelable whp.

Let $H = H_{n, cn}^k$ be the k -ary Erdős-Renyi random hypergraph with density c . By choice of c , H is not orientable whp. More strongly even, there exists $\delta = \delta(\varepsilon) > 0$ such that the largest *partial orientation*, i.e. the largest subset of the edges that can be oriented, has size $(1 - \delta)cn + o(n)$ whp, see for instance [27].

We set $\ell = \frac{k}{\delta c}$ and consider F as well as the hypergraph \tilde{F} where the vertices i and $i + n\ell$ for all $i \in \{1, \dots, (k - 1)n\}$ are merged. This “glues” the last $k - 1$ segments of F on top of the first $k - 1$ segments of F , making \tilde{F} a “seamless” version of our construction. Crucially, the *random weak limit* of \tilde{F} and H coincide, i.e. for any constant $R \in \mathbb{N}$ the distribution of the R -neighbourhood $N_{\tilde{F}}(v, R)$ of a random vertex v of \tilde{F} has the same limit (as $n \rightarrow \infty$) as the distribution of the R -neighbourhood $N_H(v, R)$ of a random vertex v of H .⁵ It now follows from [27, Theorem 4.1] that the size of the largest partial orientation of \tilde{F} is essentially also a $(1 - \delta)$ -fraction of the number of edges, namely $(1 - \delta)c\ell n + o(n)$ whp. Switching from \tilde{F} back to F can increase the size of a largest partial orientation by at most $(k - 1)n$ to $(1 - \delta + \frac{k-1}{c\ell})c\ell n + o(n) = (1 - \frac{\delta}{k})c\ell n + o(n)$ whp. Thus F is not orientable whp and therefore not peelable whp. ◀

7 Experiments

We used our hypergraphs to implement retrieval data structures and compare it to existing implementations.

A *1-bit retrieval data structure* for a universe \mathcal{U} is a pair of algorithms **construct** and **query**, where the input of **construct** is a set $S \subseteq \mathcal{U}$ of size $m = |S|$ and $f : S \rightarrow \{0, 1\}$. If **construct** succeeds, then the output is a data structure D_f such that $\text{query}(D_f, x) = f(x)$ for all $x \in S$. The output of $\text{query}(D_f, y)$ for $y \in \mathcal{U} \setminus S$ may yield an arbitrary element of $\{0, 1\}$. The interesting setting is when the data structure may only occupy $\mathcal{O}(m)$ bits. See [8, 7, 12, 21, 38].

One approach is to map each element $x \in S$ to a set $e_x \subset [N]$ via a hash function, where $N = m/c$ for some desired edge density c . One then seeks a solution $z : [N] \rightarrow \{0, 1\}$ satisfying $\bigoplus_{v \in e_x} z(v) = f(x)$ for all $x \in S$. The bit-vector z and the hash function then form D_f . A query simply evaluates the left hand side of the equation for x to recover $f(x)$. To compute z , we consider the hypergraph $H = ([N], \{e_x, x \in S\})$. A peelable vertex $v \in [N]$ only contained in one edge e_x corresponds to a variable $z(v)$ only occurring in the equation associated with x . It is thus easy to see that if H is peelable, repeated elimination and back-substitution yields z in $\mathcal{O}(m)$ time.

⁵ The common limit of the incidence graphs of \tilde{F} and H is the bipartite unimodular Galton-Watson tree described in [27, Section 4]. Standard arguments, e.g. from [24, 25] suffice to establish the identity.

■ **Table 2** Overheads and average running times per key of various practical retrieval data structures.

	Configuration	Overhead	construct [μs/key]	query [ns]
Botelho et al. [8]	$c = 0.81$	23.5%	0.32	59
⟨Fuse Graphs⟩	$c = 0.910, k = 3, \ell = 100$	12.1%	0.29	55
⟨Fuse Graphs⟩	$c = 0.960, k = 4, \ell = 200$	5.7%	0.29	60
⟨Fuse Graphs⟩	$c = 0.985, k = 7, \ell = 500$	2.7%	0.38	74
Luby et al. [28]	$c = 0.9, D = 12$	11.1%	0.79	94
Luby et al. [28]	$c = 0.99, D = 150$	1.1%	0.87	109
Genuzio et al. [21]	$c = 0.91, k = 3, C = 10^4$	10.2%	1.30	58
Genuzio et al. [21]	$c = 0.97, k = 4, C = 10^4$	3.4%	2.20	64
the authors [13]	$c = 0.9995, \ell = 16, C = 10^4$	0.25%	2.47	56

We implemented the following peeling-based variations and report results in⁶ Table 2. By the *overhead* of an implementation we mean $\frac{N'}{m} - 1$ where $N' \geq N$ is the total number of bits used, including auxiliary data structures.

Botelho et al. [8] H is a 3-ary Erdős-Renyi hypergraph with an edge density below the peelability threshold $c_3 \approx 0.818$. Construction via peeling and queries are very fast, but the overhead of 23% is sizeable (i.e. D_f occupies roughly $1.23m$ bits).

Fuse Graphs. The edges are distributed such that H is a fuse graph. Recall that the edge density is $c \frac{\ell}{\ell+k-1}$. Note that we let ℓ grow with k to keep the density close to c . We still keep ℓ in a moderate range, as our construction relies on $n \gg \ell$.

Luby et al. [28] The edges are distributed such that H is the peelable hypergraph from [28] already mentioned on page 2. To our knowledge these hypergraphs have not been considered in the context of retrieval. They seem to be particularly well suited to achieve very small overheads at the cost of larger construction and mean query times compared to our other approaches. Note that the largest edge size is $D + 4$ and the worst-case query time is therefore much larger than the reported average query time.

For reference, we also implemented two recent retrieval data structures that do not rely on peeling but solve linear systems [13, 21]. There, to counteract cubic solving time, the input is partitioned into chunks of size C . Especially [13] achieves much smaller overheads than what is feasible with peeling approaches, with the downside of being much slower and more complicated.

Overall, it seems using fuse graphs in retrieval data structures has a chance of outperforming existing approaches when moderate memory overheads of $\approx 5\%$ are acceptable.

However, more research is required to explore the complex space of possible input sizes, configurations of the data structures and trade-offs between overhead and runtime. Our implementations are configured reasonably, but arbitrary in some aspects. A full discussion is beyond the scope of this paper.

⁶ Experiments were performed on a desktop computer with an Intel® Core i7-2600 Processor @ 3.40GHz. In all cases, the data set S contains the first $m = 10^7$ URLs from the `eu-2015-host` dataset gathered by [5] with ≈ 80 bytes per key, and $f: \mathcal{U} \rightarrow \{0, 1\}$ is taken to be the parity of the string length. As hash function we used MURMURHASH3_x64_128 [2]. If more than 128 hash bits were needed, techniques resembling double-hashing were used to generate additional bits to avoid another execution of murmur. Reported query times are averages obtained by querying all elements of the data set once. They include the roughly 25 ns needed to evaluate murmur on average. The reported numbers are medians of 5 executions.

8 Conclusion

We introduced for all $k \in \mathbb{N}$ a new family of k -uniform hypergraphs where the vertex set is partitioned into a large but constant number of segments. Each edge chooses a random range of k consecutive segments and one random incidence in each of them.

While we have no asymptotic results on the resulting peelability thresholds f_k , at least for small k they are remarkably close to c_k^* with $0 \leq c_k^* - f_k \leq 10^{-5}$ for $k \in \{3, 4, 5, 6, 7\}$. In other words, f_k almost coincides with the *orientability* threshold c_k^* of Erdős-Renyi hypergraphs and significantly exceeds their peelability threshold c_k . Note that $c_k^* = 1 - (1 + o_k(1))e^{-k} \xrightarrow{k \rightarrow \infty} 1$ (see [19, page 3]) while $c_k \xrightarrow{k \rightarrow \infty} 0$ (see e.g. [35]). When plugging our hypergraphs into the retrieval framework by [8], we obtained corresponding improvements with respect to memory usage, with no discernible downsides.

Future Experiments. While our experiments on retrieval data structures are promising, it is unclear how robustly the advantages translate to other practical settings where peelable hypergraphs are used, say when implementing Invertible Bloom Lookup Tables [22]. There are hidden disadvantages of our hypergraphs not considered in this paper – for instance the number of rounds needed to peel our hypergraphs is higher, possibly hurting parallel peeling algorithms – as well as hidden advantages – peeling in external memory, a setting considered in [3], is easy due to the locality of the edges.

A Theoretical Question. Given our results, it is natural to suspect a fundamental connection between f_k and c_k^* . Quite possibly, the tiny gap that seems to remain between the values – clearly negligible from a practical perspective – is merely an artefact of the discreteness of segments in our construction.

This discreteness, while heavily used in our arguments, may in fact be dispensable. Indeed, we believe the key idea behind our hypergraphs is *limited bandwidth* where a hypergraph on vertex set $[n]$ has bandwidth at most d if each edge e satisfies $\max_{v \in e} v - \min_{v \in e} v < d$ (the incidence matrix can then be sorted to resemble a bandmatrix). Such a hypergraph can be generated by choosing for each edge a random range of d consecutive vertices and k incidences independently and uniformly at random from that range. In experiments with $k = 3$ and $d = \varepsilon n$, such hypergraphs performed similar to the hypergraphs we analysed (with $k = 3$ and $\ell \approx 1/\varepsilon$). Note that there are no discrete segments in the modified construction. It would be nice to see whether in such a variation peelability and orientability are more elegantly and more intimately linked.

References

- 1 David Aldous and J. Michael Steele. The objective method: Probabilistic combinatorial optimization and local weak convergence. In *Probability on Discrete Structures. Encyclopaedia of Mathematical Sciences (Probability Theory)*, volume 110, pages 1–72. Springer, Berlin, Heidelberg, 2004. doi:10.1007/978-3-662-09444-0_1.
- 2 Austin Appleby. MurmurHash3, 2012. URL: <https://github.com/aappleby/smhasher/blob/master/src/MurmurHash3.cpp>.
- 3 Djamel Belazzougui, Paolo Boldi, Giuseppe Ottaviano, Rossano Venturini, and Sebastiano Vigna. Cache-oblivious peeling of random hypergraphs. In *Data Compression Conference*, pages 352–361, 2014. doi:10.1109/DCC.2014.48.
- 4 Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 1970. doi:10.1145/362686.362692.

- 5 Paolo Boldi, Andrea Marino, Massimo Santini, and Sebastiano Vigna. BUBiNG: Massive crawling for the masses. In *Proc. 23rd WWW'14*, pages 227–228, 2014. doi:10.1145/2567948.2577304.
- 6 Fabiano Cupertino Botelho. *Near-Optimal Space Perfect Hashing Algorithms*. PhD thesis, Federal University of Minas Gerais, 2008. URL: <http://cmph.sourceforge.net/papers/thesis.pdf>.
- 7 Fabiano Cupertino Botelho, Rasmus Pagh, and Nivio Ziviani. Simple and space-efficient minimal perfect hash functions. In *Proc. 10th WADS*, pages 139–150, 2007. doi:10.1007/978-3-540-73951-7_13.
- 8 Fabiano Cupertino Botelho, Rasmus Pagh, and Nivio Ziviani. Practical perfect hashing in nearly optimal space. *Inf. Syst.*, pages 108–131, 2013. doi:10.1016/j.is.2012.06.002.
- 9 Julie Anne Cain, Peter Sanders, and Nicholas C. Wormald. The random graph threshold for k -orientability and a fast algorithm for optimal multiple-choice allocation. In *Proc. 18th SODA*, pages 469–476, 2007. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283433>.
- 10 Denis Xavier Charles and Kumar Chellapilla. Bloomier filters: A second look. In *Proc. 16th ESA*, 2008. doi:10.1007/978-3-540-87744-8_22.
- 11 Martin Dietzfelbinger, Andreas Goerdts, Michael Mitzenmacher, Andrea Montanari, Rasmus Pagh, and Michael Rink. Tight thresholds for cuckoo hashing via XORSAT. In *Proc. 37th ICALP (1)*, pages 213–225, 2010. doi:10.1007/978-3-642-14165-2_19.
- 12 Martin Dietzfelbinger and Rasmus Pagh. Succinct data structures for retrieval and approximate membership (extended abstract). In *Proc. 35th ICALP (1)*, pages 385–396, 2008. doi:10.1007/978-3-540-70575-8_32.
- 13 Martin Dietzfelbinger and Stefan Walzer. Constant-time retrieval with $O(\log m)$ extra bits. In *Proc. 36th STACS*, pages 24:1–24:16, 2019. doi:10.4230/LIPIcs.STACS.2019.24.
- 14 Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theor. Comput. Sci.*, 380(1-2):47–68, 2007. doi:10.1016/j.tcs.2007.02.054.
- 15 Olivier Dubois and Jacques Mandler. The 3-XORSAT threshold. In *Proc. 43rd FOCS*, pages 769–778, 2002. doi:10.1109/SFCS.2002.1182002.
- 16 David Eppstein and Michael T. Goodrich. Straggler identification in round-trip data streams via Newton’s identities and invertible Bloom filters. *IEEE Trans. on Knowl. and Data Eng.*, 23(2):297–306, 2011. doi:10.1109/TKDE.2010.132.
- 17 Daniel Fernholz and Vijaya Ramachandran. The k -orientability thresholds for $g_{n,p}$. In *Proc. 18th SODA*, pages 459–468, 2007. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283432>.
- 18 Nikolaos Fountoulakis, Megha Khosla, and Konstantinos Panagiotou. The multiple-orientability thresholds for random hypergraphs. *Combinatorics, Probability & Computing*, 25(6):870–908, 2016. doi:10.1017/S0963548315000334.
- 19 Nikolaos Fountoulakis and Konstantinos Panagiotou. Sharp load thresholds for cuckoo hashing. *Random Struct. Algorithms*, 41(3):306–333, 2012. doi:10.1002/rsa.20426.
- 20 Alan M. Frieze and Páll Melsted. Maximum matchings in random bipartite graphs and the space utilization of cuckoo hash tables. *Random Struct. Algorithms*, 41(3):334–364, 2012. doi:10.1002/rsa.20427.
- 21 Marco Genuzio, Giuseppe Ottaviano, and Sebastiano Vigna. Fast scalable construction of (minimal perfect hash) functions. In *Proc. 15th SEA*, pages 339–352, 2016. doi:10.1007/978-3-319-38851-9_23.
- 22 Michael T. Goodrich and Michael Mitzenmacher. Invertible Bloom lookup tables. In *Proc. 49th Annual Allerton Conference on Communication, Control, and Computing*, pages 792–799, 2011. doi:10.1109/Allerton.2011.6120248.
- 23 Svante Janson and Malwina J. Luczak. A simple solution to the k -core problem. *Random Struct. Algorithms*, 30(1-2):50–62, 2007. doi:10.1002/rsa.20147.

- 24 Jeong Han Kim. Poisson cloning model for random graphs. In *Proc. ICM Madrid 2006 Vol. III*, pages 873–898, 2006. URL: <https://www.mathunion.org/fileadmin/ICM/Proceedings/ICM2006.3/ICM2006.3.ocr.pdf>.
- 25 Mathieu Leconte. Double hashing thresholds via local weak convergence. In *51st Annual Allerton Conference on Communication, Control, and Computing*, pages 131–137, 2013. doi:10.1109/Allerton.2013.6736515.
- 26 Eric Lehman and Rina Panigrahy. 3.5-way cuckoo hashing for the price of 2-and-a-bit. In *Proc. 17th ESA*, pages 671–681, 2009. doi:10.1007/978-3-642-04128-0_60.
- 27 Marc Lelarge. A new approach to the orientation of random hypergraphs. In *Proc. 23rd SODA*, pages 251–264, 2012. doi:10.1137/1.9781611973099.23.
- 28 Michael Luby, Michael Mitzenmacher, Mohammad Amin Shokrollahi, and Daniel A. Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584, 2001. doi:10.1109/18.910575.
- 29 Tomasz Luczak. Size and connectivity of the k -core of a random graph. *Discrete Mathematics*, 91(1):61–68, 1991. doi:10.1016/0012-365X(91)90162-U.
- 30 Bohdan S. Majewski, Nicholas C. Wormald, George Havas, and Zbigniew J. Czech. A family of perfect hashing methods. *Comput. J.*, pages 547–554, 1996. doi:10.1093/comjnl/39.6.547.
- 31 Michael Mitzenmacher. Some open questions related to cuckoo hashing. In *Proc. 17th ESA*, 2009. doi:10.1007/978-3-642-04128-0_1.
- 32 Michael Mitzenmacher, Konstantinos Panagiotou, and Stefan Walzer. Load thresholds for cuckoo hashing with double hashing. In *16th SWAT*, pages 29:1–29:9, 2018. doi:10.4230/LIPIcs.SWAT.2018.29.
- 33 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press, New York, NY, USA, 2nd edition, 2017.
- 34 Michael Mitzenmacher and George Varghese. Biff (Bloom filter) codes: Fast error correction for large data sets. In *Proc. ISIT 2012*, pages 483–487, 2012. doi:10.1109/ISIT.2012.6284236.
- 35 Michael Molloy. Cores in random hypergraphs and boolean formulas. *Random Struct. Algorithms*, 27(1):124–135, 2005. doi:10.1002/rsa.20061.
- 36 Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004. doi:10.1016/j.jalgor.2003.12.002.
- 37 Boris Pittel and Gregory B. Sorkin. The satisfiability threshold for k -XORSAT. *Combinatorics, Probability & Computing*, 25(2):236–268, 2016. doi:10.1017/S0963548315000097.
- 38 Ely Porat. An optimal Bloom filter replacement based on matrix solving. In *Proc. 4th CSR*, pages 263–273, 2009. doi:10.1007/978-3-642-03351-3_25.
- 39 Michael Rink. Mixed hypergraphs for linear-time construction of denser hashing-based data structures. In *Proc. 39th SOFSEM*, pages 356–368, 2013. doi:10.1007/978-3-642-35843-2_31.
- 40 Stefan Walzer. Load thresholds for cuckoo hashing with overlapping blocks. In *Proc. 45th ICALP*, pages 102:1–102:10, 2018. doi:10.4230/LIPIcs.ICALP.2018.102.

Efficient Gauss Elimination for Near-Quadratic Matrices with One Short Random Block per Row, with Applications

Martin Dietzfelbinger 

Technische Universität Ilmenau, Germany
martin.dietzfelbinger@tu-ilmenau.de

Stefan Walzer 

Technische Universität Ilmenau, Germany
stefan.walzer@tu-ilmenau.de

Abstract

In this paper we identify a new class of sparse near-quadratic random Boolean matrices that have full row rank over $\mathbb{F}_2 = \{0, 1\}$ with high probability and can be transformed into echelon form in almost linear time by a simple version of Gauss elimination. The random matrix with dimensions $n(1 - \varepsilon) \times n$ is generated as follows: In each row, identify a block of length $L = O((\log n)/\varepsilon)$ at a random position. The entries outside the block are 0, the entries inside the block are given by fair coin tosses. Sorting the rows according to the positions of the blocks transforms the matrix into a kind of band matrix, on which, as it turns out, Gauss elimination works very efficiently with high probability. For the proof, the effects of Gauss elimination are interpreted as a (“coin-flipping”) variant of Robin Hood hashing, whose behaviour can be captured in terms of a simple Markov model from queuing theory. Bounds for expected construction time and high success probability follow from results in this area. They readily extend to larger finite fields in place of \mathbb{F}_2 .

By employing hashing, this matrix family leads to a new implementation of a *retrieval* data structure, which represents an arbitrary function $f: S \rightarrow \{0, 1\}$ for some set S of $m = (1 - \varepsilon)n$ keys. It requires $m/(1 - \varepsilon)$ bits of space, construction takes $\mathcal{O}(m/\varepsilon^2)$ expected time on a word RAM, while queries take $\mathcal{O}(1/\varepsilon)$ time and access only one contiguous segment of $\mathcal{O}((\log m)/\varepsilon)$ bits in the representation ($\mathcal{O}(1/\varepsilon)$ consecutive words on a word RAM). The method is readily implemented and highly practical, and it is competitive with state-of-the-art methods. In a more theoretical variant, which works only for unrealistically large S , we can even achieve construction time $\mathcal{O}(m/\varepsilon)$ and query time $\mathcal{O}(1)$, accessing $\mathcal{O}(1)$ contiguous memory words for a query. By well-established methods the retrieval data structure leads to efficient constructions of (static) perfect hash functions and (static) Bloom filters with almost optimal space and very local storage access patterns for queries.

2012 ACM Subject Classification Theory of computation \rightarrow Data structures design and analysis

Keywords and phrases Random Band Matrix, Gauss Elimination, Retrieval, Hashing, Succinct Data Structure, Randomised Data Structure, Robin Hood Hashing, Bloom Filter

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.39

Acknowledgements We are very grateful to Seth Pettie, who triggered this research by asking an insightful question regarding “one block” while discussing the two-block solution from [17]. (This discussion took place at the Dagstuhl Seminar 19051 “Data Structures for the Cloud and External Memory Data”.) Thanks are also due to the reviewers, whose comments helped to improve the presentation.



© Martin Dietzfelbinger and Stefan Walzer;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 39; pp. 39:1–39:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

1.1 Sparse Random Matrices

In this paper we introduce and study a new class of sparse random matrices over finite fields, which give rise to linear systems that are efficiently solvable with high probability¹. For concreteness and ease of notation, we describe the techniques for the field $\mathbb{F}_2 = \{0, 1\}$. (The analysis applies to larger fields as well, as will be discussed below.) A matrix A from this class has n columns and $m = (1 - \varepsilon)n$ rows for some small $\varepsilon > 0$. We always imagine that a right hand side $\vec{b} \in \{0, 1\}^m$ is given and that we wish to solve the system $A\vec{z} = \vec{b}$ for the vector of unknowns \vec{z} .

The applications (see Section 1.2) dictate that the rows of A are stochastically independent and are all chosen according to the same distribution \mathcal{R} on $\{0, 1\}^n$. Often, but not always, \mathcal{R} is the uniform distribution on some pool $R \subseteq \{0, 1\}^n$ of admissible rows. The following choices were considered in the literature.

- (1) If $R = \{0, 1\}^n$, then A has full row rank whp for any $\varepsilon = \omega(1/n)$. In fact, the probability for full row rank is > 0.28 even for $\varepsilon = 0$, see e.g. [11, 35]. Solving time is $\tilde{O}(n^3)$.
- (2) A popular choice for R is the set of vectors with 1's in precisely k positions, for constant k . Then $\varepsilon = e^{-\theta(k)}$ is sufficient for solvability whp [34]. Solving time is still $\tilde{O}(n^3)$ if Gauss elimination is used and $\mathcal{O}(n^2)$ if Wiedemann's algorithm [37] is used, but heuristics exploiting the sparsity of A help considerably [22].
- (3) In the previous setting with $k = 3$ and $\varepsilon \geq 0.19$, linear running time can be achieved with a simple greedy algorithm, since then the matrix can be brought into echelon form by row and column *exchanges* alone [7, 24, 32]. Using $k > 3$ is pointless here, as then the required value of ε increases.
- (4) Luby et al. [29, 30] study "loss-resilient codes" based on certain random bipartite graphs. Translating their considerations into our terminology shows that at the core of their construction is a distribution on $(1 - \varepsilon)n \times n$ -matrices with randomly chosen sparse rows as well. Simplifying a bit, a number $D = \mathcal{O}(1/\varepsilon)$ is chosen and a weight sequence is carefully selected that will give a row at most D many 1's and on average $\mathcal{O}(\log D)$ many 1's (in random positions). It is shown in [29, 30] that such matrices not only have full row rank with high probability, but that, as in (3), row and column *exchanges* suffice to obtain an echelon form whp. This leads to a solving time of $\mathcal{O}(n \log(1/\varepsilon))$ for the corresponding linear system.
- (5) The authors of the present work describe in a simultaneous paper [18] the construction of sparse $(1 - \varepsilon)n \times n$ matrices for very small (constant) ε , with a fixed number of 1's per row, which also allow solving the corresponding system by row and column exchanges. (While behaviour in experiments is promising, determining the behaviour of the construction for arbitrarily small ε is an open problem.)
- (6) In a recent proposal [17] by the authors of the present paper, a row $r \sim \mathcal{R}$ contains two *blocks* of $\Theta(\log n)$ random bits at random positions (block-aligned) in a vector otherwise filled with 0's. It turned out that in this case even $\varepsilon = \mathcal{O}((\log n)/n)$ will give solvability with high probability. Solution time is again about cubic (Gauss) resp. quadratic (Wiedemann), with heuristic pre-solvers cushioning the blow partly in practice.

Motivated by the last construction, we propose an even simpler choice for the distribution \mathcal{R} : A row r consists of 0's except for one randomly placed block of some length L , which consists of random bits. It turns out that $L = \mathcal{O}((\log n)/\varepsilon)$ is sufficient to achieve solvability

¹ Events occur "with high probability (whp)" if they occur with probability $1 - \mathcal{O}(m^{-1})$.

with high probability. The L -bit block fits into $\mathcal{O}(1)$ memory words as long as ε is constant. Our main technical result (Theorem 2) is that the resulting random matrix has full row rank whp. Moreover, if this is the case then sorting the rows by starting points of the blocks followed by a simple version of Gauss elimination produces an echelon form of the matrix and a solution to the linear system. The expected number of field operations is $\mathcal{O}(nL/\varepsilon)$, which translates into expected running time $\mathcal{O}(n/\varepsilon^2)$ on a word RAM. For the proof, we establish a connection to a particular version of Robin Hood hashing, whose behaviour in turn can be understood by reducing it to a well-known situation in queuing theory. (A detailed sketch of the argument is provided in Section 2.1.)

To our knowledge, this class of random matrices has not been considered before. However, *deterministic* versions of matrices similar to these random ones have been thoroughly studied in the last century, for both infinite and finite fields. Namely, sorting the rows of our matrices yields matrices that with high probability resemble *band matrices*, where the nonzero entries in row i are within a restricted range around column $\lfloor i/(1 - \varepsilon) \rfloor$. In the study of band matrices one usually has $\varepsilon = 0$ and assumes that the matrix is nonsingular. Seemingly the best known general upper time bound for the number of field operations needed for solving band quadratic systems with bandwidth L are $\mathcal{O}(nL^{\omega-1}) = \mathcal{O}(n((\log n)/\varepsilon)^{\omega-1})$, where ω is the matrix multiplication exponent, see [20, 23, 33].

1.2 Retrieval

One motivation for studying random systems as described above comes from data structures for solving the *retrieval problem*, which can be described as follows: Some “*universe*” \mathcal{U} of possible keys is given, as is a function $f: S \rightarrow W$, where $S \subseteq \mathcal{U}$ has finite size m and $W = \{0, 1\}^r$ for some $r \geq 1$. A *retrieval data structure* [6, 10, 15, 35] makes it possible to recover $f(x)$ quickly for arbitrary given $x \in S$. We do not care what the result is when $x \notin S$, which makes the retrieval situation different from a dictionary, where the question “ $x \in S$?” must also be decided. A retrieval data structure consists of

- an algorithm **construct**, which takes f as a list of pairs (and maybe some parameters) as input and constructs an object DS_f , and
- an algorithm **query**, which on input $x \in \mathcal{U}$ and DS_f outputs an element of W , with the requirement that $\text{query}(\text{DS}_f, x) = f(x)$ for all $x \in S$.

The essential performance parameters of a retrieval data structure are:

- the space taken up by DS_f (ideally $(1 + \varepsilon)m$ bits of memory for some small $\varepsilon > 0$),
- the running time of **construct** (ideally $\mathcal{O}(m)$), and
- the running time of **query** (ideally a small constant in the worst case, possibly dependent on ε , and good cache behaviour).

In this paper we concentrate on the case most relevant in practice, namely the case of small constant r , in particular on² $r = 1$.

A standard approach is as follows [6, 10, 15, 35]. Let $f: S \rightarrow \{0, 1\}$ be given and let $n = m/(1 - \varepsilon)$ for some $\varepsilon > 0$. Use hashing to construct a mapping $\text{row}: \mathcal{U} \rightarrow \{0, 1\}^n$ such that $(\text{row}(x))_{x \in S}$ is (or behaves like) a family of independent random variables drawn from a suitable distribution \mathcal{R} on $\{0, 1\}^n$. Consider the linear system $(\langle \text{row}(x), \vec{z} \rangle = f(x))_{x \in S}$. In case the vectors $\text{row}(x)$, $x \in S$, are linearly independent, this system is solvable for \vec{z} .

² Every solution for this case gives a solution for larger r as well, with a slowdown not larger than r . In our case, this slowdown essentially only affects queries, not construction, since the Gauss elimination based algorithm can trivially be extended to simultaneously handle r right hand sides $\vec{b}_1, \dots, \vec{b}_r$ and produce r solution vectors $\vec{z}_1, \dots, \vec{z}_r$. This change slows down **construct** by a factor of $1 + r/L = 1 + \mathcal{O}(\varepsilon r / \log n)$.

Solve the system and store the bit vector \vec{z} of n bits (and the hash function used) as DS_f . Evaluation is by $\text{query}(\text{DS}_f, x) = \langle \text{row}(x), \vec{z} \rangle$, for $x \in \mathcal{U}$. The running time of **construct** is essentially the time for solving the linear system, and the running time for **query** is the time for evaluating the inner product.

A common and well-explored trick for reducing the construction time [5, 16, 35, 22] is to split the key set into “chunks” of size $\Theta(C)$ for some suitable C and constructing separate retrieval structures for the chunks. The price for this is twofold: In queries, one more hash function must be evaluated and the proper part of the data structure has to be located; regarding storage space one needs an array of $\Omega(m/C)$ pointers. In this paper, we first concentrate on a “pure” construction. The theoretical improvements possible by applying the splitting technique will be discussed briefly in Section 4. The splitting technique is also used in experiments for our construction in Section 5 to keep the block length small. In this context it will also be noted the related “split-and-share” technique from [16, 19] can be used to get rid of the assumption that fully random hash functions are available for free.

Our main result regarding the retrieval problem follows effortlessly from the analysis of the new random linear systems (formally stated as Theorem 2).

► **Theorem 1.** *Let \mathcal{U} be a universe. Assume the context of a word RAM with oracle access to fully random hash functions on \mathcal{U} . Then for any $\varepsilon > 0$ there is a retrieval data structure such that for all $S \subseteq \mathcal{U}$ of size m*

- (i) *construct succeeds with high probability.*
- (ii) *construct has expected running time $\mathcal{O}(\frac{m}{\varepsilon^2})$.*
- (iii) *The resulting data structure DS_f occupies at most $(1 + \varepsilon)m$ bits.*
- (iv) *query has running time $\mathcal{O}(\frac{1}{\varepsilon})$ and accesses $\mathcal{O}(\frac{1}{\varepsilon})$ consecutive words in memory.*

1.3 Machine Model and Notation

For a positive integer k we denote $\{1, \dots, k\}$ by $[k]$. The number m always denotes the size of a domain – the number of keys to hash, the size of a function for retrieval or the number of rows of a matrix. A (small) real number $\varepsilon > 0$ is also given. The number n denotes the size of a range. We usually have $m = (1 - \varepsilon)n$. In asymptotic considerations we always assume that ε is constant and m and n tend to ∞ , so that for example the expression $\mathcal{O}(n/\varepsilon)$ denotes a function that is bounded by cm/ε for a constant c , for all m bigger than some $m(\varepsilon)$. By $\langle \vec{y}, \vec{z} \rangle$ we denote the inner product of two vectors \vec{y} and \vec{z} . As our computational model we adopt the word RAM with memory words comprising $\Omega(\log m)$ bits, in which an operation on a word takes constant time. In addition to AC_0 instructions we will need the **PARITY** of a word as an elementary operation. For simplicity we assume this can be carried out in constant time, which certainly is realistic for standard word lengths like 64 or 128. In any case, as the word lengths used are never larger than $\mathcal{O}(\log m)$, one could tabulate the values of **PARITY** for inputs of size $\frac{1}{2} \log m$ in a table of size $\mathcal{O}(\sqrt{m} \log m)$ bits to achieve constant evaluation time for inputs comprising a constant number of words.

1.4 Techniques Used

We use *coupling* of random variables X and Y (or of processes $(X_i)_{i \geq 1}$ and $(Y_i)_{i \geq 1}$). By this we mean that we exhibit a single probability space on which X and Y (or $(X_i)_{i \geq 1}$ and $(Y_i)_{i \geq 1}$) are defined, so that there are interesting *pointwise* relations between them, like $X \leq Y$, or $X_i \leq Y_i + a$ for all $i \geq 1$, for a constant a . Sometimes these relations hold only conditioned on some (large) part of the probability space. We will make use of the following

observation. If we have random variables U_0, \dots, U_k with couplings, i.e. joint distributions, of $U_{\ell-1}$ and U_ℓ , for $1 \leq \ell \leq k$, then there is a common probability space on which all these random variables are defined and the joint distribution of $U_{\ell-1}$ and U_ℓ is as given.³

2 Random Band Systems that Can be Solved Quickly

The main topic of this paper are matrices generated by the following random process. Let $0 < \varepsilon < 1$ and $n \in \mathbb{N}$. For a number $m = (1 - \varepsilon)n$ of rows and some number $L \geq 1$ we consider a matrix $A = (a_{ij})_{i \in [m], j \in [n+L-1]}$ over the field \mathbb{F}_2 , chosen at random as follows. For each row $i \in [m]$ a *starting position* $s_i \in [n] = \{1, \dots, n\}$ is chosen uniformly at random. The entries a_{ij} , $s_i \leq j < s_i + L$ form a *block* of fully random bits, all other entries in row i are 0.

In this section we show that for proper choices of the parameters such a random matrix will have full row rank and the corresponding systems $A\vec{z} = \vec{b}$ will be solvable very efficiently whp. Before delving into the technical details, we sketch the main ideas of the proof.

2.1 Proof Sketch

As a starting point, we formulate a simple algorithm, a special version of Gaussian elimination, for solving linear systems $A\vec{z} = \vec{b}$ as just described. We first sort the rows of A by the starting position of their block. The resulting matrix resembles a band matrix, and we apply standard Gaussian elimination to it, treating the rows in order of their starting position. Conveniently, there is no “proliferation of 1’s”, i.e. we never produce a 1-entry outside of any row’s original block. In the round for row i , the entries a_{ij} for $j = s_i, \dots, s_i + L - 1$ are scanned. If column j has been previously chosen as pivot then $a_{ij} = 0$. Otherwise, a_{ij} is a random bit. While this bit may depend in a complex way on the original entries of rows $1, \dots, i$ (apart from position (i, j)), for the analysis we may simply imagine that a_{ij} is only chosen now by flipping a fair coin. This means that we consider eligible columns from left to right, and the first j for which the coin flip turns up 1 becomes the pivot column for row i . This view makes it possible to regard choosing pivot columns for the rows as probabilistically equivalent to a slightly twisted version of Robin Hood hashing. Here this means that m keys x_1, \dots, x_m with random hash values in $\{1, \dots, n + L - 1\}$ are given and, in order of increasing hash values, are inserted in a linear probing fashion into a table with positions $1, \dots, n + L - 1$ (meaning that for x_i cells s_i, s_{i+1}, \dots are inspected). The twist is that whenever a key probes an empty table cell flipping a fair coin decides whether it is placed in the cell or has to move on to the next one. The resulting position of key x_i is the same as the position of the pivot for row i . As is standard in the precise analysis of linear probing hashing we switch perspective and look at the process from the point of view of cells $1, 2, \dots, n, \dots, n + L - 1$. Associated with position (“time”) j is the set of keys that probe cell j (the “queue”), and the quantity to study is the length of this queue. It turns out that the average queue length determines the overall cost of the row additions, and that the probability for the maximum queue length to become too large is decisive for bounding the success probability of the Gaussian elimination process. The first and routine step in

³ We do not prove this formally, since arguments like this belong to basic probability theory or measure theory. The principle used is that the pairwise couplings give rise to conditional expectations $\mathbb{E}(U_\ell \mid U_{\ell-1})$. Arguing inductively, given a common probability space for $U_1, \dots, U_{\ell-1}$ and $\mathbb{E}(U_\ell \mid U_{\ell-1})$, one can obtain a common probability space for U_1, \dots, U_ℓ so that $(U_1, \dots, U_{\ell-1})$ is distributed as before and $\mathbb{E}(U_\ell \mid U_1, \dots, U_{\ell-1}) = \mathbb{E}(U_\ell \mid U_{\ell-1})$. – This is practically the same as the standard argument that shows that a sequence of conditional expectations gives rise to a corresponding Markov chain on a joint probability space.

the analysis of the queue length is to “Poissonise” arrivals such that the evolution of the queue length becomes a Markov chain. A second step is needed to deal with the somewhat annoying possibility that in a cell all keys that are eligible for this cell reject it because of their coin flips. We end up with a standard queue (an “M/D/1 queue” in Kendall notation) and can use existing results from queuing theory to read off the bounds regarding the queue length needed to complete the analysis.

The following subsections give the details.

2.2 A Simple Gaussian Solver

We now describe the algorithm to solve linear systems involving the random matrices described above. This is done by a variant of Gauss elimination, which will bring the matrix into echelon form (up to leaving out inessential column exchanges) and then apply back substitution.

Given a random matrix $A = (a_{ij})_{i \in [m], j \in [n+L-1]}$ as defined above, with blocks of length L starting at positions s_i , for $i \in [m]$, as well as some $\vec{b} \in \{0, 1\}^m$, we wish to find a solution \vec{z} to the system $A\vec{z} = \vec{b}$. Consider algorithm SGAUSS (Algorithm 1). If A has linearly independent rows, it will return a solution \vec{z} and produce intermediate values $(\text{piv}_i)_{i \in [m]}$. (These will be important only in the analysis of the algorithm.) If the rows of A are linearly dependent, the algorithm will fail.

■ **Algorithm 1** A simple Gaussian solver.

```

1 Algorithm SGAUSS( $A = (a_{ij})_{i \in [m], j \in [n+L-1]}$ ,  $(s_i)_{i \in [m]}$ ,  $\vec{b} \in \{0, 1\}^m$ ):
2   sort the rows of the system  $(A, \vec{b})$  by  $s_i$  (in time  $\mathcal{O}(m)$ )
3   relabel such that  $s_1 \leq s_2 \leq \dots \leq s_m$ 
4    $\text{piv}_1, \text{piv}_2, \dots, \text{piv}_m \leftarrow 0$ 
5   for  $i = 1, \dots, m$  do
6     for  $j = s_i, \dots, s_i + L - 1$  do } // search for leftmost 1 in row  $i$ . Can be done
7     if  $a_{ij} = 1$  then } // in time  $\mathcal{O}(L/\log m)$  on a word RAM.
8        $\text{piv}_i \leftarrow j$ 
9       for  $i'$  with  $i' > i \wedge s_{i'} \leq \text{piv}_i$  do
10        if  $a_{i', \text{piv}_i} = 1$  then
11           $a_{i'} \leftarrow a_{i'} \oplus a_i$  // row addition (= subtraction)
12           $b_{i'} \leftarrow b_{i'} \oplus b_i$ 
13        break
14     if  $\text{piv}_i = 0$  // row  $i$  is 0
15     then return FAILURE
    // back substitution:
16    $\vec{z} \leftarrow \vec{0}$ 
17   for  $i = m, \dots, 1$  do
18      $z_{\text{piv}_i} \leftarrow \langle \vec{z}, a_i \rangle \oplus b_i$  // note:  $a_{ij} = 0$  for  $j$  outside of  $\{s_i, \dots, s_i + L - 1\}$ 
19   return  $\vec{z}$  // solution to  $A\vec{z} = \vec{b}$ 

```

Algorithm SGAUSS starts by sorting the rows of the system (A, \vec{b}) by their starting positions s_i in linear time, e.g. using counting sort [13, Chapter 8.2]. We suppress the resulting permutation in the notation, assuming $s_1 \leq s_2 \leq \dots \leq s_m$. Rows are then processed sequentially. When row i is treated, its leftmost 1-entry is found, if possible, and

the corresponding column index is called the *pivot* piv_i of row i . Row additions are used to eliminate 1-entries from column piv_i in subsequent rows. Note that this operation never produces nonzero entries outside of any row's original block, i.e. for no row i are there ever any 1's outside of the positions $\{s_i, \dots, s_i + L - 1\}$. To see this, we argue inductively on the number of additions performed. Assume $i > 1$ and row i' with $i' < i$ is added to row i . By choice of $\text{piv}_{i'}$ and the induction hypothesis, nonzero entries of row i' can reside only in positions $\text{piv}_{i'}, \dots, s_{i'} + L - 1$. Again by induction and since row i contains a 1 in position $\text{piv}_{i'}$, we have $s_i \leq \text{piv}_{i'}$; moreover we have $s_{i'} + L - 1 \leq s_i + L - 1$, due to sorting. Thus, row i' contains no 1's outside of the block of row i and the row addition maintains the invariant.

If an all-zero row is encountered, the algorithm fails (and returns FAILURE). This happens if and only if the rows of A are linearly dependent⁴. Otherwise we say that the algorithm succeeds. In this case a solution \vec{z} to $A\vec{z} = \vec{b}$ is obtained by back-substitution.

It is not hard to see that the expected running time of SGAUSS is dominated by the expected cost of row additions.

The proof of the following statement, presented in the rest of this section, is the main technical contribution of this paper.

► **Theorem 2.** *There is some $L = \mathcal{O}((\log m)/\varepsilon)$ such that a run of SGAUSS on the random matrix $A = (a_{ij})_{i \in [m], j \in [n+L-1]}$ and an arbitrary right hand side $\vec{b} \in \{0, 1\}^m$ succeeds whp. The expected number of row additions is $\mathcal{O}(m/\varepsilon)$. Each row addition involves entries inside one block and takes time $\mathcal{O}(1/\varepsilon)$ on a word RAM.*

2.3 Coin-Flipping Robin Hood Hashing

Let $\{x_1, \dots, x_m\} \subseteq \mathcal{U}$ be some set of *keys* to be stored in a hash table T . Each key x_i has a uniformly random *hash value* $h_i \in [n]$. An (injective) placement of the keys in T fulfils the *linear probing* requirement if each x_i is stored in a cell $T[\text{pos}_i]$ with $\text{pos}_i \geq h_i$ and all cells $T[j]$ for $h_i \leq j < \text{pos}_i$ are non-empty. In *Robin Hood hashing* there is the additional requirement that $h_i > h_{i'}$ implies $\text{pos}_i > \text{pos}_{i'}$. Robin Hood hashing is interesting because it minimises the variance of the displacements $\text{pos}_i - h_i$. It has been studied in detail in several papers [9, 14, 25, 27, 36].

Given the hash values $(h_i)_{i \in [m]}$, a placement of the keys obeying the Robin Hood linear probing conditions can be obtained as follows: Insert the keys in the order of increasing hash values, by the usual linear probing insertion procedure, which probes (i.e. inspects) cells $T[h_i], T[h_i + 1], \dots$ until the first empty cell is found, and places x_i in this cell. We consider a slightly “broken” variation of this method, which sometimes delays placements. In the placing procedure for x_i , when an empty cell $T[j]$ is encountered, it is decided by flipping a fair coin whether to place x_i in cell $T[j]$ or move on to the next cell. (No problem is caused by the fact that the resulting placement violates the Robin Hood requirement and even the linear probing requirement, since the hash table is only used as a tool in our analysis.) For this insertion method we assume we have an (idealised) unbounded array $T[1, 2, \dots]$. The position in which key x_i is placed is called pos_i . At the end the algorithm itself checks whether any of the displacements $\text{pos}_i - h_i$ is larger than L , in which case it reports FAILURE.⁵ Algorithm 2 gives a precise description of this algorithm, which we term CFRH.

⁴ Depending on \vec{b} , the system $A\vec{z} = \vec{b}$ may still be solvable. We will not pursue this.

⁵ The reason we postpone checking for FAILURE until the very end of the execution is that it is technically convenient to have the values $(\text{pos}_i)_{i \in [m]}$ even if failure occurs.

■ **Algorithm 2** The *Coin-Flipping Robin Hood hashing* algorithm. Without the condition “ $\text{coinFlip}() = 1$ ” it would compute a Robin Hood placement with maximum displacement L , if one exists.

```

1 Algorithm CFRH ( $\{x_1, \dots, x_m\} \subseteq \mathcal{U}$ ):
2   sort  $x_1, \dots, x_m$  by hash value  $h_1, \dots, h_m$ 
3   relabel such that  $h_1 \leq \dots \leq h_m$ 
4    $T \leftarrow [\perp, \perp, \dots]$  // empty array, “ $\perp$ ” means “undefined”
5    $\text{pos}_1, \dots, \text{pos}_m \leftarrow 0$ 
6   for  $i = 1, \dots, m$  do
7     for  $j = h_i, h_i + 1, \dots$  do
8       if  $T[j] = \perp \wedge \text{coinFlip}() = 1$  (“HEADS”) then
9          $\text{pos}_i \leftarrow j$ 
10         $T[j] \leftarrow x_i$ 
11        break
12  if  $\exists i \in [m] : \text{pos}_i - h_i \geq L$  then return FAILURE
13  return  $T$ 

```

2.4 Connection between SGAUSS and CFRH

We now establish a close connection between the behaviour of algorithms SGAUSS and CFRH, thus reducing the analysis of SGAUSS to that of CFRH. The algorithms have been formulated in such a way that some structural similarity is immediate. A run of SGAUSS on a matrix with random starting positions $(s_i)_{i \in [m]}$ and random entries yields a sequence of pivots $(\text{piv}_i)_{i \in [m]}$; a run of CFRH on a key set with random hash values $(h_i)_{i \in [m]}$ performing random coin flips yields a sequence of positions $(\text{pos}_i)_{i \in [m]}$. We will see that the distributions of $(\text{piv}_i)_{i \in [m]}$ and $(\text{pos}_i)_{i \in [m]}$ are essentially the same and that moreover two not so obvious parameters of the two random processes are closely connected. For this, we will show that outside the FAILURE events we can use the probability space underlying algorithm SGAUSS to describe the behaviour of algorithm CFRH. This yields a coupling of the involved random processes.

The first step is to identify $s_i = h_i$ for $i \in [m]$ (both sequences are assumed to be sorted and then renamed). The connection between pos_i and piv_i is achieved by connecting the coin flips of CFRH to certain events in applying SGAUSS to matrix A . We construct this correspondence by induction on i . Assume rows $1, \dots, i-1$ have been treated, x_1, \dots, x_{i-1} have been placed, and $\text{piv}_{i'} = \text{pos}_{i'}$ for all $1 \leq i' < i$.

Now row a_i (transformed by previous row additions) is treated. It contains a 0 in columns that were previously chosen as pivots, so possible candidates for piv_i are only indices from $J_i := \{s_i, \dots, s_i + L - 1\} \setminus \{\text{piv}_1, \dots, \text{piv}_{i-1}\}$. For each $j \in J_i$, the initial value of a_{ij} was a random bit. The bits added to a_{ij} in rounds $1, \dots, i-1$ are determined by the original entries of rows $1, \dots, i-1$ alone. We order the entries of J_i as $j^{(1)} < j^{(2)} < \dots < j^{(|J_i|)}$. Then, conditioned on all random choices in rows $1, \dots, i-1$ of A , the current values $a_{i,j^{(1)}}, \dots, a_{i,j^{(k)}}$ still form a sequence of fully random bits. We use these random bits to run round i of CFRH, in which x_i is placed. Since each cell can only hold one key, and by excluding runs where finally FAILURE is declared, we may focus on the empty cells with indices in $\{h_i, \dots, h_i + L - 1\} \setminus \{\text{pos}_1, \dots, \text{pos}_{i-1}\} = \{s_1, \dots, s_i + L - 1\} \setminus \{\text{piv}_1, \dots, \text{piv}_{i-1}\} = J_i$. We use (the current value) a_{ij} as the value of the coin flip for cell j , for $j = j^{(1)}, j^{(2)}, \dots, j^{(|J_i|)}$. The minimal j in this sequence (if any) with $a_{ij} = 1$ equals piv_i and pos_i . If all these bits are 0, algorithm SGAUSS will fail immediately, and key x_i will be placed in a cell $T[j]$ with $j \geq h_i + L$, so CFRH will eventually fail as well.

Thus we have established that the random variables needed to run algorithm CFRH (outside of FAILURE) can be taken to belong to the probability space defined by $(s_i)_{i \in [m]}$ and the entries in the blocks of A for algorithm SGAUSS, so that (outside of FAILURE) the random variables pos_i and piv_i are the same. In the following lemma we state this connection as Claim (i). In addition, we consider other random variables central for the analysis to follow. First, we define the *height* of position $j \in [n + L - 1]$ in the hash table as

$$H_j := \#\{i \in [m] \mid h_i \leq j < \text{pos}_i\}.$$

This is the number of keys probing table cell j without being placed in it, either because the cell is occupied or because it is rejected by the coin flip. Claim (ii) in the next lemma shows that $\sum_{j \in [n+L-1]} H_j$ essentially determines the running time of SGAUSS, so that we can focus on bounding $(H_j)_{j \in \mathbb{N}}$ from here on. Further, with Claim (iii), we get a handle on the question how large we have to choose L in order to keep the failure probability small.

► **Lemma 3.** *With the coupling just described, we get*

- (i) SGAUSS succeeds iff CFRH succeeds. On success we have $\text{piv}_i = \text{pos}_i$ for all $i \in [m]$.
- (ii) A successful run of SGAUSS performs at most $\sum_{j \in [n+L-1]} H_j$ row additions.
- (iii) Conditioned on the event $\max_{j \in [n]} H_j \leq L - 2 \log m$, the algorithms succeed whp.

Proof. (ii) (Note that a similar statement with a different proof can be found in [26, Lemma 2.1].) Consider the sets $\text{Add} := \{(i, i') \in [m]^2 \mid \text{SGAUSS adds row } i \text{ to row } i'\}$ and $\text{Displ} := \{(i, j) \in [m] \times [n + L - 1] \mid h_i \leq j < \text{pos}_i\}$. Since H_j simply counts the pairs $(i, j) \in \text{Displ}$ with $i \in [m]$, we have $|\text{Displ}| = \sum_{j \in [n+L-1]} H_j$. To prove the claim we exhibit an injection from Add into Displ .

Assume $(i, i') \in \text{Add}$. If $\text{pos}_i < \text{pos}_{i'}$, we map (i, i') to (i', pos_i) . This is indeed an element of Displ , since $h_{i'} \leq \text{piv}_i = \text{pos}_i < \text{pos}_{i'}$ (if piv_i were smaller than $s_{i'}$, row i would not be added to row i'). On the other hand, if $\text{pos}_i > \text{pos}_{i'}$, we map (i, i') to $(i, \text{pos}_{i'})$. This is in Displ since $h_i = s_i \leq s_{i'} \leq \text{pos}_{i'} < \text{pos}_i$ (recall that rows are sorted by starting position).

The mapping is injective since from the image of $(i, i') \in \text{Add}$ we can recover the set $\{i, i'\}$ with the help of the injective mapping $i \mapsto \text{pos}_i$, $i \in [m]$. The fact that $i < i'$ fixes the ordering in the pair.

(iii) In CFRH, for an arbitrary $i \in [m]$ consider the state before key x_i probes its first position $j := h_i$. Any previous key $x_{i'}$ with $i' < i$ has a hash value $h_{i'} \leq h_i$. Hence it either was inserted in a cell $j' < j$ or it has probed cell j . Since at most H_j keys have probed cell j , at most H_j positions in $T[j, \dots, j + L - 1]$ are occupied and at least $2 \log m$ are free. The probability that x_i is not placed in this region is therefore at most $2^{-2 \log m} = m^{-2}$. By the union bound we obtain a failure probability of $\mathcal{O}(1/m)$. ◀

2.5 Bounding Heights in CFRH by a Markov Chain

Lemma 3 tells us that we must analyse the heights in the hashing process CFRH. In this subsection, we use “Poissonisation” of the hashing positions to majorise the heights in CFRH by a Markov chain, i.e. a process that is oblivious to the past, apart from the current height. Poissonisation is a common step in the analysis of linear probing hashing, see e.g. [36]. Further, we wish to replace randomized placement by deterministic placement: Whenever a key is available for a position, one is put there (instead of flipping coins for all available keys). By this, the heights may decrease, but only by a bounded amount whp. The details of these steps are given in this subsection.

In analysing CFRH (without regard for the event FAILURE), it is inconvenient that the starting positions h_i are determined by random choices with subsequent sorting. Position j is hit by a number of keys given by a binomial distribution $\text{Bin}(m, \frac{1}{n})$ with expectation $\frac{m}{n} = 1 - \varepsilon$, but there are dependencies. We approximate this situation by ‘‘Poissonisation’’ [31, Sect. 5.4]. Here this means that we assume that cell $j \in [n]$ is hit by k_j keys, independently for $j = 1, \dots, m$, where $k_j \sim \text{Po}(1 - \varepsilon')$ is Poisson distributed, for $\varepsilon' = \varepsilon/2$. Then the total number $m' = \sum_{j \in [n]} k_j$ of keys is distributed as $m' \sim \text{Po}((1 - \varepsilon')n)$. Given k_1, \dots, k_n , we can imagine we have m' keys with nondecreasing hash values $(h_i)_{i \in [m']}$, and we can apply algorithm CFRH to obtain key positions $(\text{pos}'_i)_{i \in [m']}$ in $\{1, 2, \dots\}$ and cell heights $(H'_j)_{j \geq 1}$.

Conveniently, with Poissonisation, the heights $(H'_j)_{j \in [n]}$ turn out to form a Markov chain. This can be seen as follows. Recall that H'_{j-1} is the number of keys probing cell $j - 1$ without being placed there. Hence the number of keys probing cell j is $H'_{j-1} + k_j$. One of these keys will be placed in cell j , unless $H'_{j-1} + k_j$ coin flips all yield 0, so if $g_j \sim \text{Geom}(\frac{1}{2})$ is a random variable with geometric distribution with parameter $\frac{1}{2}$ (number of fair coin flips needed until the first 1 appears) and b_j is the indicator function $\mathbb{1}_{\{g_j > H'_{j-1} + k_j\}}$, we have $H'_j = H'_{j-1} + k_j - 1 + b_j$. (Note that the case $H'_{j-1} + k_j = 0$ is treated correctly by this description. Conditioned on $H'_{j-1} + k_j$, the value b_j is a Bernoulli variable.) The Markov property holds since H'_j depends only on H'_{j-1} and the two ‘‘fresh’’ random variables k_j and g_j .

The following lemma allows us to shift our attention from (H_j) to (H'_j) .

► **Lemma 4.** *Let $m = (1 - \varepsilon)n$ and $m' \sim \text{Po}((1 - \varepsilon')m)$ for $\varepsilon' = \varepsilon/2$. There is a coupling between an ordinary run of CFRH (with m , n and H_j) and a Poissonised run (with m' , n and H'_j) such that conditioned on the high probability event $E_{\geq m} = \{m' \geq m\}$ we have $H'_j \geq H_j$ for all $j \in [n + L - 1]$.*

Proof. Because ε and $\varepsilon' = \varepsilon/2$ are constants, the event $E_{\geq m}$ has indeed high probability, as can be seen by well-known concentration bounds for the Poisson distribution (e.g. [31, Th. 5.4]). For $m_0 \geq m$ fixed the distribution of the number of hits in the cells in $T[1, \dots, n]$ conditioned on $\{m' = m_0\}$ is the same as what we get by throwing m_0 balls randomly into n bins [31, Th. 5.6]. Thus, we may assume the Poissonised run has to deal with the m keys of the ordinary run plus $m' - m$ additional keys with random hash values in $[n]$. We apply algorithm CFRH to both inputs. After sorting, the new keys are inserted in some interleaved way with the ordinary keys. Now if one of the ordinary keys x probes an empty cell $T[j]$, we use the same coin flip in both runs to decide whether to place it there; for the probing of the additional keys we use new, independent coin flips. With this coupling it is clear that for all ordinary keys x the displacement ‘‘(position of x) – (hash value of x)’’ in the Poissonised run is at least as big as in the ordinary run. As the additional keys can only increase heights, $H'_j \geq H_j$ follows. ◀

As a further simplification, we eliminate the geometrically distributed variable g_j and the derived variable b_j in the Markov chain $(H'_j)_{j \geq 0}$. For this, let $(X_j)_{j \geq 0}$ be the Markov chain defined as

$$X_0 := 0 \quad \text{and} \quad X_j := \max(0, X_{j-1} + d_j - 1) \quad \text{for } j \geq 1, \quad (1)$$

where $d_j \sim \text{Po}(1 - \varepsilon'/2)$ are independent random variables.

► **Lemma 5.** *There is a coupling between $(X_j)_{j \geq 0}$ and $(H'_j)_{j \geq 0}$ such that $X_j + \log(4/\varepsilon') \geq H'_j$ for all $j \in [n + L - 1]$.*

Proof. Assume wlog that $\log(1/\varepsilon')$ is an integer. Let $b'_j \sim \text{Po}(\varepsilon'/2)$ be a random variable on the same probability space as g_j such that $g_j > \log(4/\varepsilon')$ implies $b'_j \geq 1$. This is possible because

$$\Pr[g_j > \log(4/\varepsilon')] = 2^{-\log(4/\varepsilon')} = \varepsilon'/4 \leq 1 - e^{-\varepsilon'/2} = \Pr[b'_j \geq 1].$$

We then define $d_j := k_j + b'_j$ which gives $d_j \sim \text{Po}(1 - \varepsilon'/2)$. Proceeding by induction, and using (1), we can define $(X_j)_{j \geq 0}$ and $(H'_j)_{j \geq 0}$ on a common probability space. Then we check $X_j + \log(4/\varepsilon') \geq H'_j$, also by induction: In the case $H'_{j-1} + k_j \leq \log(4/\varepsilon')$ we simply get

$$X_j + \log(4/\varepsilon') \geq \log(4/\varepsilon') \geq H'_{j-1} + k_j \geq H'_{j-1} + k_j + b_j - 1 = H'_j.$$

Otherwise we can use the inequality $b_j = \mathbb{1}_{\{g_j > H'_{j-1} + k_j\}} \leq \mathbb{1}_{\{g_j > \log(4/\varepsilon')\}} \leq b'_j$ to obtain

$$\begin{aligned} X_j + \log(4/\varepsilon') &\geq X_{j-1} + d_j - 1 + \log(4/\varepsilon') \stackrel{(\text{Ind.Hyp.})}{\geq} H'_{j-1} + d_j - 1 \\ &= H'_{j-1} + k_j + b'_j - 1 \geq H'_{j-1} + k_j + b_j - 1 = H'_j. \end{aligned} \quad \blacktriangleleft$$

2.6 Enter Queuing Theory

It turns out that, in essence, the behaviour of the Markov chain $(X_j)_{j \geq 0}$ has been studied in the literature under the name “M/D/1 queue”, which is Kendall notation [28] for queues with “Markovian arrivals, Deterministic service times and 1 server”. We will exploit what is known about this simple queuing situation in order to finish our analysis.

Formally, an M/D/1 queue is a Markov process $(Z_t)_{t \in \mathbb{R}_{\geq 0}}$ in continuous time and discrete space $\mathbb{N}_0 = \{0, 1, 2, \dots\}$. The random variable Z_t is usually interpreted as the number of *customers* waiting in a FIFO queue at time $t \in \mathbb{R}_{\geq 0}$. Initially the queue is empty ($Z_0 = 0$). Customers arrive independently, i.e. arrivals are determined by a Poisson process with a rate we set to $\rho = 1 - \varepsilon'/2$ (which implies that the number of customers arriving in any fixed time interval of length 1 is $\text{Po}(\rho)$ -distributed). The *server* requires one time unit to process a customer which means that if $t \in \mathbb{R}_{\geq 0}$ is the time of the first arrival, then customers will leave the queue at times $t + 1, t + 2, \dots$ until the queue is empty again.

Now consider the discretisation $(Z_j)_{j \in \mathbb{N}_0}$ of the M/D/1 queue. For $j \geq 1$, the number d_j of arrivals in between two observations Z_{j-1} and Z_j has distribution $d_j \sim \text{Po}(\rho)$, and one customer was served in the meantime if and only if $Z_{j-1} > 0$. We can therefore write

$$Z_j = \begin{cases} d_j & \text{if } Z_{j-1} = 0, \\ Z_{j-1} + d_j - 1 & \text{if } Z_{j-1} > 0. \end{cases}$$

By reusing the variables $(d_j)_{j \geq 1}$ that previously occurred in the definition of $(X_j)_{j \geq 0}$, we already established a coupling between the processes $(X_j)_{j \geq 0}$ and $(Z_j)_{j \geq 0}$. A simple induction suffices to show

$$X_j = \max(0, Z_j - 1) \text{ for all } j \geq 0. \quad (2)$$

Intuitively, the server in the X -process is ahead by one customer because customers are processed at integer times “just in time for the observation”.

The following results are known in queuing theory:

► **Fact 1.**

(i) The average number of customers in the Z -queue at time $t \in \mathbb{R}_{\geq 0}$ is

$$\mathbb{E}[Z_t] \leq \lim_{\tau \rightarrow \infty} \mathbb{E}[Z_\tau] = \rho + \frac{1}{2} \left(\frac{\rho^2}{1 - \rho} \right) = \Theta(1/\varepsilon).$$

(Precise values are known even for general service-time distributions, see [12, Chapter 5.4].)

(ii) [21, Prop 3.4] We have the following tail bound for the event $\{Z_t > k\}$ for any $k \in \mathbb{N}$:

$$\Pr[Z_t > k] \leq \lim_{\tau \rightarrow \infty} \Pr[Z_\tau > k] = e^{-k \cdot \Theta(\varepsilon)}, \text{ for all } t \geq 0.$$

2.7 Putting the Pieces Together

We now have everything in place to prove Theorem 2 regarding solving our linear systems.

Proof of Theorem 2. By the observation made in Section 1.4, we may assume that the random variables $(H_j)_{j \in [n+L-1]}$, $(H'_j)_{j \in [n+L-1]}$, $(X_j)_{j \geq 0}$ and $(Z_j)_{j \geq 0}$ and the three corresponding couplings are realized on one common probability space.

By Fact 1(ii) it is possible to choose $L = \Theta((\log m)/\varepsilon)$ while guaranteeing $\Pr[Z_j > L/2] = \mathcal{O}(m^{-2})$ for all $j \geq 0$.

By the choice of L and the union bound, the event $E_{\max Z} = \{\forall j \in [n+L-1]: Z_j \leq L/2\}$ occurs whp. Conditioned on $E_{\max Z}$ and the high probability event $E_{\geq m}$ from Lemma 4 we have

$$H_j \stackrel{\text{Lem. 4}}{\leq} H'_j \stackrel{\text{Lem. 5}}{\leq} X_j + \log(4/\varepsilon') \stackrel{\text{Eq. 2}}{\leq} Z_j + \log(4/\varepsilon') \stackrel{E_{\max Z}}{\leq} L/2 + \log(4/\varepsilon') \leq L - 2 \log m.$$

By using Lemma 3(iii) we conclude that SGAUSS succeeds with high probability.

Along similar lines we get, for each $j \in [n+L-1]$:

$$\begin{aligned} \mathbb{E}[H_j] &\stackrel{\text{Lem. 4}}{\leq} \mathbb{E}[H'_j \mid E_{\geq m}] \leq \frac{1}{\Pr[E_{\geq m}]} \mathbb{E}[H'_j] \stackrel{\text{Lem. 5}}{\leq} \frac{1}{\Pr[E_{\geq m}]} \mathbb{E}[X_j + \log(4/\varepsilon')] \\ &\stackrel{\text{Eq. 2}}{\leq} \frac{1}{\Pr[E_{\geq m}]} \mathbb{E}[Z_j + \log(4/\varepsilon')] \stackrel{\text{Fact 1(i)}}{\leq} \frac{1}{\Pr[E_{\geq m}]} (\mathcal{O}(1/\varepsilon) + \log(4/\varepsilon')) = \mathcal{O}(1/\varepsilon). \end{aligned}$$

By Lemma 3(ii) the expected number of row additions performed by a successful run of SGAUSS is therefore at most $\mathbb{E}[\sum_{j \in [n+L-1]} H_j] = \mathcal{O}(m/\varepsilon)$. Since unsuccessful runs happen with probability $\mathcal{O}(1/m)$ and can perform at most mL additions (each row can only be the target of L row additions), the overall expected number of additions is not skewed by unsuccessful runs, hence is also in $\mathcal{O}(m/\varepsilon)$. This finishes the proof of Theorem 2. ◀

► **Remark.** The analysis described in this section works in exactly the same way if instead of \mathbb{F}_2 a larger finite field \mathbb{F} is used. A row in the random matrix is determined by a random starting position and a block of L random elements from \mathbb{F} . A row operation in the Gaussian elimination now consists of a division, a multiplication of a block with a scalar and a row addition. The running time of the algorithm will increase at least by a factor of $\log(|\mathbb{F}|)$ (the bitlength of a field element), and further increases depend on how well word parallelism in the word RAM can be utilized for operations like row additions, scalar multiplications and scalar products. (In [22], efficient methods are described for the field of three elements.) The queue length will become a little smaller, but not significantly, since even the M/D/1 queue with arrivals with a Poisson($1 - \varepsilon$) distribution will lead to average queue length $\Theta(1/\varepsilon)$.

► **Remark.** An interesting question was raised by a reviewer of the submission: Is anything gained if we fix the first bit of each block to be 1? When checking our analysis for this case we see that this 1-bit need not survive previous row operations. However, such a change does improve success probabilities in the Robin Hood insertion procedure. If a key x_i finds cell h_i empty, it occupies this cell, without a coin being flipped. From the point of view of the queues, we see that now the derived variable b_j in Section 2.5 is 1 if $k_j > 0$ and geometrically distributed only if $k_j = 0$. As in the preceding remark, this brings the process closer to the M/D/1 queue with arrivals with a $\text{Poisson}(1 - \varepsilon)$ distribution and deterministic service time 1, but the average queue length remains $\Theta(1/\varepsilon)$. Still, it may be interesting to check by experiments if improvements result by this change.

3 A New Retrieval Data Structure

With Theorem 2 in place we are ready to carry out the analysis of the retrieval data structure based on the new random matrices as described in Section 1.2. The proof of Theorem 1 is more or less straightforward.

Proof of Theorem 1. Denote the m elements of S by x_1, \dots, x_m , let $n = \frac{1}{1-\varepsilon}m$, $L = \Theta(\frac{\log m}{\varepsilon})$ the number from Theorem 2 and $h: \mathcal{U} \rightarrow [n] \times \{0, 1\}^L$ a fully random hash function. For construct, we associate the values $(s_i, p_i) := h(x_i)$ with each x_i for $i \in [m]$ and interpret them as a random band matrix $A = (a_{ij})_{i \in [m], j \in [n+L-1]}$, where for all $i \in [m]$ row a_i contains the pattern p_i starting at position s_i and 0's everywhere else. Moreover, let $\vec{b} \in \{0, 1\}^m$ be the vector with entries $b_i = f(x_i)$ for $i \in [m]$. We call SGAUSS (Algorithm 1) with inputs A and \vec{b} , obtaining (in case of success) a solution $\vec{z} \in \{0, 1\}^{n+L-1}$ of $A\vec{z} = \vec{b}$. The retrieval data structure is simply $\text{DS}_f = \vec{z}$.

By Theorem 2 construct succeeds whp⁶ (establishing (i)) and performs $\mathcal{O}(m/\varepsilon)$ row additions. Since additions affect only $L = \mathcal{O}(\frac{\log m}{\varepsilon})$ consecutive bits, and since a word RAM can deal with $\mathcal{O}(\log m)$ bits at once, a single row addition costs $\mathcal{O}(1/\varepsilon)$ time, leading to total expected running time $\mathcal{O}(m/\varepsilon^2)$ (which establishes (ii)).

The data structure $\text{DS}_f = \vec{z}$ occupies exactly $\frac{1}{1-\varepsilon}m + L - 1 < (1 + 2\varepsilon)m$ bits. Replacing ε with $\varepsilon/2$ yields the literal result (iii).

To evaluate $\text{query}(\text{DS}_f, y)$ for $y \in \mathcal{U}$, we compute $(s_y, p_y) = h(y)$ and the scalar product $b_y = \langle \vec{z}[s_y \dots s_y+L-1], p_y \rangle := \bigoplus_{j=1}^L \vec{z}_{s_y+j-1} \cdot p_{y_j}$. By construction, this yields $b_i = f(x_i)$ in the case that $y = x_i$. To obtain (iv), observe that the scalar product of two binary sequences of length $L = \mathcal{O}(\log(n)/\varepsilon)$ can be computed using $\mathcal{O}(1/\varepsilon)$ bit parallel AND and XOR operations, as well as a single PARITY operation on $\mathcal{O}(\log m)$ bits, which can be assumed to be available in constant time. ◀

► **Remark.** As the proof of Theorem 2 remains valid for arbitrary fixed finite fields in place of \mathbb{F}_2 , the same is true for Theorem 1. This is relevant for the compact representation of functions with small ranges like [3], where binary encoding of single symbols implies extra space overhead. Such functions occur in data structures for perfect hash functions [7, 22].

⁶ If success with probability 1 is desired, then in case the construction fails with hash function $h_0 = h$, we just restart the construction with different hash functions h_1, h_2, \dots . In this setup, DS_f must also contain the seed $s \in \mathbb{N}_0$ identifying the first hash function h_s that led to success.

4 Input Partitioning

We examine the effect of a simple trick to improve construction and query times of our retrieval data structure. We partition the input into *chunks* using a “first-level hash function” and construct a separate retrieval data structure for each chunk. Using this with chunk size $C = m^\varepsilon$ will reduce the time bounds for construction and query by a factor of ε . The main reason for this is that we can use smaller block sizes L , which in turn makes row additions and inner products cheaper. Note that the idea is not new. Partitioning the input has previously been applied in the context of retrieval to reduce construction times, especially when “raw” construction times are superlinear [15, 22, 35] or when performance in external memory settings is an issue [3, 7]. Partitioning also allows us to get rid of the full randomness assumption, which is interesting from a theoretical point of view [7, 19, 16].

► **Remark.** The reader should be aware that the choice $C = m^\varepsilon$, which is needed to obtain a speedup of $1/\varepsilon$, is unlikely to be a good choice in practice and that this improvement only works for unrealistically large m . Namely, we use that $\frac{\log m}{m^\varepsilon} \ll \varepsilon$ for sufficiently large m . While the left term is indeed $o(1)$ and the right a constant, even for moderate values of ε like 0.05 implausibly large values of m are needed to satisfy the weaker requirement $\frac{\log m}{m^\varepsilon} < \varepsilon$. In this sense, Theorem 6 taken literally is of purely theoretical value. Still, the general idea is sound and it can give improvements in practice when partitioning less aggressively, say with $C \approx \sqrt{m}$. For example, the good running times reported in Section 5 are only possible with this splitting approach.

► **Theorem 6.** *The result of Theorem 1 can be strengthened in the following ways.*

- (i) *The statements of Theorem 1 continue to hold without the assumption of fully random hash functions being available for free.*
- (ii) *The expected construction time is $\mathcal{O}(m/\varepsilon)$ (instead of $\mathcal{O}(m/\varepsilon^2)$).*
- (iii) *The expected query time is $\mathcal{O}(1)$ (instead of $\mathcal{O}(1/\varepsilon)$). Queries involve accessing a (small) auxiliary data structure, so technically not all required data is “consecutive in memory”.*

Proof Sketch. Let $C = m^\varepsilon$ be the *desired chunk size*. In [7, Section 4] it is described in detail how a splitting function can be used to obtain chunks that have size within a constant factor of C with high probability, and how fully random hash functions on each individual chunk can be provided by a randomized auxiliary structure \mathcal{H} that takes only $o(m)$ space. New functions can be generated by switching to new seeds. (This construction is a variation of what is described in [16, 19].) This fully suffices for our purposes. We construct an individual retrieval data structure for each chunk with $L = \mathcal{O}(\frac{\log C}{\varepsilon}) = \mathcal{O}(\log m)$. Such a construction succeeds in expected time $\mathcal{O}(C/\varepsilon)$ with probability $1 - \mathcal{O}(1/C)$. In case the construction fails for a chunk, it is repeated with a different seed. At the end we save the concatenation of all m/C retrieval data structures, the data structure \mathcal{H} and an auxiliary array. This array contains, for each chunk, the offset of the corresponding retrieval data structure in the concatenation and the seed of the hash function used for the chunk. It is easy to check that the size of all auxiliary data is asymptotically negligible.

The total expected construction time is $\mathcal{O}((m/C) \cdot C/\varepsilon) = \mathcal{O}(m/\varepsilon)$, and since $L = \mathcal{O}(\log m)$, a retrieval query can be evaluated in constant time. ◀

► **Remark.** The construction from [30] described in item (4) in the list in Section 1.1 can also be transformed in a retrieval data structure. (This does not seem to have been explored up to now.) The expected running time for **construct** is $\mathcal{O}(m \log(1/\varepsilon))$ (better than our $\mathcal{O}(m/\varepsilon)$), the expected running time for **query** is $\mathcal{O}(\log(1/\varepsilon))$, with $\mathcal{O}(\log(1/\varepsilon))$ random accesses in

memory. (Worst case is $\mathcal{O}(1/\varepsilon)$.) In our preliminary experiments, see Section 5, for $m = 10^7$, both construction and query times of our construction seem to be able to compete well with the construction following [30].

5 Experiments

We implemented our retrieval data structure following the approach explained in the proof of Theorem 6, except that we used `MurmurHash3` [1] for all hash functions. This is a heuristic insofar as we depart from the full randomness assumption of Theorem 1. We report⁷ running times and space overheads in Table 1, with the understanding that a retrieval data structure occupying N bits of memory in total and accommodating m keys has overhead $\frac{N}{m} - 1$. Concerning the choice of parameters, $L = 64$ has practical advantages on a 64-bit machine and $C = 10^4$ seems to go well with it experimentally. As $\varepsilon \in \{7\%, 5\%, 3\%\}$ decreases, the measured construction time increases as would be expected. This is partly due to the higher number of row additions in successful constructions, but also due to an increased probability for a chunk's construction to fail, which prompts a restart for that chunk with a different seed. Note that, in our implementation, querying an element in a chunk with non-default seed also prompts an additional hash function evaluation.

Competing Implementations. For comparison, we implemented the retrieval data structures from [7, 17, 22] and the one arising from the construction in [29]. (The number D in Table 1 is the maximum number of 1's in a row; the average is then $\Theta(\log D)$.)

In [7], the rows of the linear systems contain three 1's in independent and uniformly random positions. If the number of columns is $n = m/(1 - \varepsilon)$ for $\varepsilon > 18.2\%$, the system can be solved in linear time by row and column *exchanges* alone. Compared to that method, we achieve smaller overheads at similar running times.

The approaches from [22] and [17] are different in that they construct linear systems that require cubic solving time with Gaussian elimination. This is counteracted by partitioning the input into chunks as well as by a heuristic *LazyGauss*-phase of the solver that eliminates many variables before the Method of Four Russians [2] is used on what remains. Construction times are higher than ours, but the tiny space overhead achieved in [17] is beyond the reach of our approach. The systems considered in [22] resemble those in [7], except at higher densities. The systems studied in [17] resemble our systems, except with *two* blocks of random bits per row instead of one.

We remark that our approach is easier to implement than those of [17, 22] but more difficult than that of [7].

6 Conclusion

This paper studies the principles of solving linear systems given by a particular kind of sparse random matrices, with one short random block per row, in a random position. The proof works by the point of view from Gaussian elimination to Robin Hood hashing and then to queuing theory. It might be interesting to find a direct, simpler proof for the main

⁷ Experiments were performed on a desktop computer with an Intel® Core i7-2600 Processor @ 3.40GHz. Following [22], we used as data set S the first $m = 10^7$ URLs from the `eu-2015-host` dataset gathered by [4] with ≈ 80 bytes per key. As hash function we used `MURMURHASH3_X64_128` [1]. Reported query times are averages obtained by querying all elements of the data set once and include the evaluation of `murmur`, which takes about 25 ns on average. The reported figures are medians of 5 executions.

■ **Table 1** Space overhead and running times per key of some practical retrieval data structures.

	Configuration	Overhead	construct [$\mu\text{s}/\text{key}$]	query [ns]
[7]	$\varepsilon = 19\%$	23.5%	0.32	59
$\langle\text{NEW}\rangle$	$\varepsilon = 7\%, L = 64, C = 10^4$	8.8%	0.24	52
$\langle\text{NEW}\rangle$	$\varepsilon = 5\%, L = 64, C = 10^4$	6.5%	0.27	54
$\langle\text{NEW}\rangle$	$\varepsilon = 3\%, L = 64, C = 10^4$	4.3%	0.43	61
[29]	$c = 0.9, D = 12$	11.1%	0.79	94
[29]	$c = 0.99, D = 150$	1.1%	0.87	109
[22]	$\varepsilon = 9\%, k = 3, C = 10^4$	10.2%	1.30	58
[22]	$\varepsilon = 3\%, k = 4, C = 10^4$	3.4%	2.20	64
[17]	$\varepsilon = 0.05\%, \ell = 16, C = 10^4$	0.25%	2.47	56

theorem. Preliminary experiments concerning an application with retrieval data structures are promising. The most intriguing property is that evaluation of a retrieval query requires accessing only one (short) block in memory.

The potential of the construction in practice should be explored more fully and systematically, taking all relevant parameters like block size and chunk size into consideration. Constructions of perfect hash functions like in [7, 22] or Bloom filters that combine perfect hashing with fingerprinting [8, 15] might profit from our construction.

References

- 1 Austin Appleby. MurmurHash3, 2012. URL: <https://github.com/aappleby/smhasher/blob/master/src/MurmurHash3.cpp>.
- 2 Gregory V. Bard. *Algebraic Cryptanalysis*, chapter The Method of Four Russians, pages 133–158. Springer US, Boston, MA, 2009. doi:10.1007/978-0-387-88757-9_9.
- 3 Djamel Belazzougui, Paolo Boldi, Giuseppe Ottaviano, Rossano Venturini, and Sebastiano Vigna. Cache-oblivious peeling of random hypergraphs. In *Proc. DCC'14*, pages 352–361, 2014. doi:10.1109/DCC.2014.48.
- 4 Paolo Boldi, Andrea Marino, Massimo Santini, and Sebastiano Vigna. BUBiNG: Massive crawling for the masses. In *Proc. 23rd WWW'14*, pages 227–228. ACM, 2014. doi:10.1145/2567948.2577304.
- 5 Fabiano C. Botelho, Yoshiharu Kohayakawa, and Nivio Ziviani. A practical minimal perfect hashing method. In *Proc. 4th WEA*, pages 488–500, 2005. doi:10.1007/11427186_42.
- 6 Fabiano C. Botelho, Rasmus Pagh, and Nivio Ziviani. Simple and space-efficient minimal perfect hash functions. In *Proc. 10th WADS*, pages 139–150, 2007. doi:10.1007/978-3-540-73951-7_13.
- 7 Fabiano C. Botelho, Rasmus Pagh, and Nivio Ziviani. Practical perfect hashing in nearly optimal space. *Inf. Syst.*, 38(1):108–131, 2013. doi:10.1016/j.is.2012.06.002.
- 8 Andrei Z. Broder and Michael Mitzenmacher. Network applications of Bloom filters: A survey. *Internet Mathematics*, 2003. doi:10.1080/15427951.2004.10129096.
- 9 Pedro Celis, Per-Åke Larson, and J. Ian Munro. Robin Hood hashing. In *Proc. 26th FOCS*, pages 281–288, 1985. doi:10.1109/SFCS.1985.48.
- 10 Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal. The Bloomier filter: An efficient data structure for static support lookup tables. In *Proc. 15th SODA*, pages 30–39, 2004. URL: <http://dl.acm.org/citation.cfm?id=982792.982797>.
- 11 Colin Cooper. On the rank of random matrices. *Random Struct. Algor.*, 16(2):209–232, 2000. doi:10.1002/(SICI)1098-2418(200003)16:2<209::AID-RSA6>3.0.CO;2-1.

- 12 Robert B. Cooper. *Introduction to Queueing Theory*. Elsevier/North-Holland, 2nd edition, 1981. URL: http://www.cse.fau.edu/~bob/publications/IntroToQueueingTheory_Cooper.pdf.
- 13 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.
- 14 Luc Devroye, Pat Morin, and Alfredo Viola. On worst-case Robin Hood hashing. *SIAM J. Comput.*, 33(4):923–936, 2004. doi:10.1137/S0097539702403372.
- 15 Martin Dietzfelbinger and Rasmus Pagh. Succinct data structures for retrieval and approximate membership (extended abstract). In *Proc. 35th ICALP (1)*, pages 385–396, 2008. doi:10.1007/978-3-540-70575-8_32.
- 16 Martin Dietzfelbinger and Michael Rink. Applications of a splitting trick. In *Proc. 36th ICALP (1)*, pages 354–365, 2009. doi:10.1007/978-3-642-02927-1_30.
- 17 Martin Dietzfelbinger and Stefan Walzer. Constant-time retrieval with $O(\log m)$ extra bits. In *Proc. 36th STACS*, pages 24:1–24:16, 2019. doi:10.4230/LIPIcs.STACS.2019.24.
- 18 Martin Dietzfelbinger and Stefan Walzer. Dense peelable random uniform hypergraphs. In *Proc. 27th ESA*, pages 38:1–38:16, 2019. doi:10.4230/LIPIcs.ESA.2019.38.
- 19 Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theor. Comput. Sci.*, 380(1-2):47–68, 2007. doi:10.1016/j.tcs.2007.02.054.
- 20 Wayne Eberly. On efficient band matrix arithmetic. In *Proc. 33rd FOCS*, pages 457–463, 1992. doi:10.1109/SFCS.1992.267806.
- 21 Regina Egorova, Bert Zwart, and Onno Boxma. Sojourn time tails in the M/D/1 processor sharing queue. *Probab. Eng. Inf. Sci.*, 20:429–446, 2006. doi:10.1017/S0269964806060268.
- 22 Marco Genuzio, Giuseppe Ottaviano, and Sebastiano Vigna. Fast scalable construction of (minimal perfect hash) functions. In *Proc. 15th SEA*, pages 339–352, 2016. doi:10.1007/978-3-319-38851-9_23.
- 23 Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.
- 24 George Havas, Bohdan S. Majewski, Nicholas C. Wormald, and Zbigniew J. Czech. Graphs, hypergraphs and hashing. In *Proc. 19th WG*, pages 153–165, 1993. doi:10.1007/3-540-57899-4_49.
- 25 Svante Janson. Individual displacements for linear probing hashing with different insertion policies. *ACM Trans. Algorithms*, 1(2):177–213, 2005. doi:10.1145/1103963.1103964.
- 26 Svante Janson. Individual displacements in hashing with coalesced chains. *Comb. Probab. Comput.*, 17(6):799–814, 2008. doi:10.1017/S0963548308009395.
- 27 Svante Janson and Alfredo Viola. A unified approach to linear probing hashing with buckets. *Algorithmica*, 75(4):724–781, 2016. doi:10.1007/s00453-015-0111-x.
- 28 David G. Kendall. Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain. *Ann. Math. Statist.*, 24(3):338–354, September 1953. doi:10.1214/aoms/1177728975.
- 29 Michael Luby, Michael Mitzenmacher, Mohammad Amin Shokrollahi, and Daniel A. Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584, 2001. doi:10.1109/18.910575.
- 30 Michael Luby, Michael Mitzenmacher, Mohammad Amin Shokrollahi, Daniel A. Spielman, and Volker Stemann. Practical loss-resilient codes. In *Proc. 29th STOC*, pages 150–159, 1997. doi:10.1145/258533.258573.
- 31 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.
- 32 Michael Molloy. Cores in random hypergraphs and boolean formulas. *Random Struct. Algorithms*, 27(1):124–135, 2005. doi:10.1002/rsa.20061.
- 33 Victor Y. Pan, Isdor Sobze, and Antoine Atinkpahoun. On parallel computations with banded matrices. *Inf. Comput.*, 120(2):237–250, 1995. doi:10.1006/inco.1995.1111.

- 34 Boris Pittel and Gregory B. Sorkin. The satisfiability threshold for k -XORSAT. *Comb. Probab. Comput.*, 25(2):236–268, 2016. doi:10.1017/S0963548315000097.
- 35 Ely Porat. An optimal Bloom filter replacement based on matrix solving. In *Proc. 4th CSR*, pages 263–273, 2009. doi:10.1007/978-3-642-03351-3_25.
- 36 Alfredo Viola. Exact distribution of individual displacements in linear probing hashing. *ACM Trans. Algorithms*, 1(2):214–242, 2005. doi:10.1145/1103963.1103965.
- 37 Douglas H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory*, 32(1):54–62, 1986. doi:10.1109/TIT.1986.1057137.

Greedy Strategy Works for k -Center Clustering with Outliers and Coreset Construction

Hu Ding

School of Computer Science and Technology,
University of Science and Technology of China, He Fei, China
huding@ustc.edu.cn

Haikuo Yu

School of Computer Science and Technology,
University of Science and Technology of China, He Fei, China
yhk7786@mail.ustc.edu.cn

Zixiu Wang

School of Computer Science and Technology,
University of Science and Technology of China, He Fei, China
wzx2014@mail.ustc.edu.cn

Abstract

We study the problem of k -center clustering with outliers in arbitrary metrics and Euclidean space. Though a number of methods have been developed in the past decades, it is still quite challenging to design quality guaranteed algorithm with low complexity for this problem. Our idea is inspired by the greedy method, Gonzalez’s algorithm, for solving the problem of ordinary k -center clustering. Based on some novel observations, we show that this greedy strategy actually can handle k -center clustering with outliers efficiently, in terms of clustering quality and time complexity. We further show that the greedy approach yields small coreset for the problem in doubling metrics, so as to reduce the time complexity significantly. Our algorithms are easy to implement in practice. We test our method on both synthetic and real datasets. The experimental results suggest that our algorithms can achieve near optimal solutions and yield lower running times comparing with existing methods.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases k -center clustering, outliers, coreset, doubling metrics, random sampling

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.40

Funding *Hu Ding*: this work was supported in part by NSF through grant CCF-1656905.

Acknowledgements The authors want to thank the anonymous reviewers for their helpful comments and suggestions for improving the paper.

1 Introduction

Clustering is one of the most fundamental problems in data analysis [25]. Given a set of elements, the goal of clustering is to partition the set into several groups based on their similarities or dissimilarities. Several clustering models have been extensively studied, such as k -center, k -median, and k -means clustering [9]. In reality, datasets often are noisy and contain outliers. Moreover, outliers could seriously affect the final results in data analysis [14]. Clustering with outliers can be viewed as a generalization of ordinary clustering problems; however, the existence of outliers makes the problems to be much more challenging.

We focus on the problem of *k -center clustering with outliers* in this paper. Given a metric space with n vertices and a pre-specified number of outliers $z < n$, the problem is to find k balls to cover at least $n - z$ vertices and minimize the maximum radius of the balls. The problem also can be defined in Euclidean space so that the cluster centers can be any points in the space (i.e., not restricted to be selected from the input points). The



© Hu Ding, Haikuo Yu, and Zixiu Wang;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 40; pp. 40:1–40:16
Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2-approximation algorithms for ordinary k -center clustering (without outliers) were given in [18, 22], and it was proved that any approximation ratio lower than 2 implies $P = NP$. A 3-approximation algorithm for k -center clustering with outliers in arbitrary metrics was proposed by Charikar et al. [15]; for the problem in Euclidean space, their approximation ratio becomes 4. A following streaming $(4 + \epsilon)$ -approximation algorithm was proposed by McCutchen and Khuller [33]. Recently, Chakrabarty et al. [13] proposed a 2-approximation algorithm for metric k -center clustering with outliers (but it is unclear of the resulting approximation ratio for the problem in Euclidean space). Existing algorithms often have high time complexities. For example, the complexities of the algorithms in [15, 33] are $O(kn^2 \log n)$ and $O(\frac{1}{\epsilon}(kzn + (kz)^2 \log \Phi))$ respectively, where Φ is the ratio of the optimal radius to the smallest pairwise distance among the vertices; the algorithm in [13] needs to solve a complicated model of linear programming and the exact time complexity is not provided. The coreset based idea of Badoiu et al. [7] needs to enumerate a large number of possible cases and also yields a high complexity. Several distributed algorithms for k -center clustering with outliers were proposed recently [12, 19, 30, 32]; most of these distributed algorithms, to our best knowledge, rely on the sequential algorithm [15].

In this paper, we aim to design quality guaranteed algorithm with low complexity for the problem of k -center clustering with outliers. Our idea is inspired by the greedy method from Gonzalez [18] for solving ordinary k -center clustering. Based on some novel insights, we show that this greedy method also works for the problem with outliers (Section 2). Our approach can achieve the approximation ratio 2 with respect to the clustering cost (i.e., the radius); moreover, the time complexity is linear in the input size. Charikar et al. [16] showed that if more than z outliers are allowed to remove, the random sampling technique can be applied to reduce the data size for metric k -center clustering with outliers. Recently, Huang et al. [23] showed a similar result for instances in Euclidean space (and they name the sample as “robust coreset”). In Section 2.3, we prove that the sample size of [23] can be further reduced.

We also consider the problem in doubling metrics, motivated by the fact that many real-world datasets often manifest low intrinsic dimensions [8]. For example, image sets usually can be represented in low dimensional manifold though the Euclidean dimension of the image vectors can be very high. “Doubling dimension” is widely used for measuring the intrinsic dimensions of datasets [35] (the formal definition is given in Section 1.1). Rather than assuming the whole (X, d) has a low doubling dimension, we only assume that **the inliers of the given data have a low doubling dimension $\rho > 0$** . We do not have any assumption on the outliers; namely, the outliers can scatter arbitrarily in the space. We believe that this assumption captures a large range of high dimensional instances in reality.

With the assumption, we show that our approach can further improve the clustering quality. In particular, the greedy approach is able to construct a coreset for the problem of k -center clustering with outliers; as a consequence, the time complexity can be significantly reduced if running existing algorithms on the coreset (Section 3). *Coreset* construction is a technique for reducing data size so as to speedup the algorithms for many optimization problems; we refer the reader to the surveys [5, 34] for more details. The size of our coreset is $2z + O((2/\mu)^\rho k)$, where μ is a small parameter measuring the quality of the coreset; the construction time is $O((\frac{2}{\mu})^\rho kn)$. Note that z and k are often much smaller than n in practice; the coefficient 2 of z actually can be further reduced to be arbitrarily close to 1, by increasing the coefficient of the second term $(2/\mu)^\rho k$. Moreover, our coreset is a natural “composable coreset” [24] which could be potentially applied to distributed clustering with outliers. Very recently, Ceccarello et al. [12] also provided a coreset for k -center clustering

with z outliers in doubling metrics, where their size is $T = O((k+z)(\frac{24}{\mu})^\rho)$ with $O(nT)$ construction time. Thus our result is a significant improvement in terms of coreset size and construction time. Huang et al. [23] considered the coreset construction for k -median/means clustering with outliers in doubling metrics, however, their method cannot be extended to the case of k -center. Aghamolaei and Ghodsi [2] considered the coreset construction in doubling metrics for ordinary k -center clustering without outliers.

Our proposed algorithms are easy to implement in practice. To study the performance of our algorithms, we test them on both synthetic and real datasets in Section 4. The experimental results suggest that our method outperforms existing methods in terms of clustering quality and running time. Also, the running time can be significantly reduced via building coreset where the clustering quality can be well preserved simultaneously.

Due to the space limit, some details are omitted here, and we refer the reader to the full version of our paper.

1.1 Preliminaries

We consider the problem of k -center with outliers in arbitrary metrics and Euclidean space \mathbb{R}^D . Let (X, d) be a metric, where X contains n vertices and $d(\cdot, \cdot)$ is the distance function; with a slight abuse of notation, we also use the function d to denote the shortest distance between two subsets $X_1, X_2 \subseteq X$, i.e., $d(X_1, X_2) = \min_{p \in X_1, q \in X_2} d(p, q)$. We assume that the distance between any pair of vertices in X is given in advance; for the problem in Euclidean space, it takes $O(D)$ time to compute the distance between any pair of points. Below, we introduce several important definitions that are used throughout the paper.

► **Definition 1** (k -Center Clustering with Outliers). *Given a metric (X, d) with two positive integers k and $z < n$, k -center clustering with outliers is to find a subset $X' \subseteq X$, where $|X'| \geq n - z$, and k centers $\{c_1, \dots, c_k\} \subseteq X$, such that $\max_{p \in X'} \min_{1 \leq j \leq k} d(p, c_j)$ is minimized. If given a set P of n points in \mathbb{R}^D , the problem is to find a subset $P' \subseteq P$, where $|P'| \geq n - z$, and k centers $\{c_1, \dots, c_k\} \subset \mathbb{R}^D$, such that $\max_{p \in P'} \min_{1 \leq j \leq k} \|p - c_j\|$ is minimized.*

Note. For the sake of convenience, we describe the following definitions only in terms of metric space. In fact, the definitions can be easily modified for the problem in Euclidean space.

In this paper, we always use X_{opt} , a subset of X with size $n - z$, to denote the subset yielding the optimal solution. Also, let $\{C_1, \dots, C_k\}$ be the k clusters forming X_{opt} , and the resulting clustering cost be r_{opt} ; that is, each C_j is covered by an individual ball with radius r_{opt} .

Usually, optimization problems with outliers are challenging to solve. Thus we often relax our goal and allow to miss a little more outliers in practice. Actually the same relaxation idea has been adopted by a number of works on clustering with outliers before [3, 16, 23, 30].

► **Definition 2** ($(k, z)_\epsilon$ -Center Clustering). *Let (X, d) be an instance of k -center clustering with z outliers, and $\epsilon \geq 0$. $(k, z)_\epsilon$ -center clustering is to find a subset X' of X , where $|X'| \geq n - (1 + \epsilon)z$, such that the corresponding clustering cost of Definition 1 on X' is minimized.*

(i) *Given a set A of cluster centers ($|A|$ could be larger than k), the resulting clustering cost,*

$$\min \left\{ \max_{p \in X'} \min_{c \in A} d(p, c) \mid X' \subseteq X, |X'| \geq n - (1 + \epsilon)z \right\} \quad (1)$$

is denoted by $\phi_\epsilon(X, A)$.

- (ii) If $|A| = k$ and $\phi_\epsilon(X, A) \leq \alpha r_{opt}$ with $\alpha > 0^1$, it is called an α -approximation. Moreover, if $|A| = \beta k$ with $\beta > 1$, it is called an (α, β) -approximation.

Obviously, the problem in Definition 1 is a special case of $(k, z)_\epsilon$ -center clustering with $\epsilon = 0$. Further, Definition 1 and 2 can be naturally extended to **weighted case**: each vertex p has a non-negative weight w_p and the total weight of outliers should be equal to z ; the distance $d(p, c_j)$ in the objective function is replaced by $w_p \cdot d(p, c_j)$. Then we have the following definition of coreset.

► **Definition 3 (Coreset).** Given a small parameter $\mu \in (0, 1)$ and an instance (X, d) of k -center clustering with z outliers, a set $S \subseteq X$ is called a μ -coreset of X , if each vertex of S is assigned a non-negative weight and $\phi_0(S, H) \in (1 \pm \mu)\phi_0(X, H)$ for any set $H \subseteq X$ of k vertices.

Given a large-scale instance (X, d) , we can run existing algorithm on its coreset S to compute an approximate solution for X ; if $|S| \ll n$, the resulting running time can be significantly reduced. Formally, we have the following claim.

▷ **Claim 4.** If the set H yields an α -approximation of the μ -coreset S , it yields an $\alpha \times \frac{1+\mu}{1-\mu}$ -approximation of X .

As mentioned before, we also consider the case with low doubling dimension. Roughly speaking, doubling dimension describes the expansion rate of the metric. For any $p \in X$ and $r \geq 0$, we use $Ball(p, r)$ to denote the ball centered at p with radius r .

► **Definition 5 (Doubling Dimension).** The doubling dimension of a metric (X, d) is the smallest number $\rho > 0$, such that for any $p \in X$ and $r \geq 0$, $X \cap Ball(p, 2r)$ is always covered by the union of at most 2^ρ balls with radius r .

2 Algorithms for $(k, z)_\epsilon$ -Center Clustering

For the sake of completeness, let us briefly introduce the algorithm of [18] for ordinary k -center clustering first. Initially, it arbitrarily selects a vertex from X , and iteratively selects the following $k - 1$ vertices, where each j -th step ($2 \leq j \leq k$) chooses the vertex having the largest minimum distance to the already selected $j - 1$ vertices; finally, each input vertex is assigned to its nearest neighbor of these selected k vertices. It can be proved that this greedy strategy results in a 2-approximation of k -center clustering; the algorithm also works for the problem in Euclidean space and results in the same approximation ratio. In this section, we show that a modified version of Gonzalez's algorithm yields approximate solutions for $(k, z)_\epsilon$ -center clustering.

In Section 2.1 and 2.2, we present our results for metric k -center with outliers. Actually, it is easy to see that Algorithm 1 and 2 yield the same approximation ratios if the input instance is a set of points in Euclidean space (the analysis is almost identical, and we omit the details due to the space limit); only the running times are different, since it takes $O(D)$ time to compute distance between two points in \mathbb{R}^D .

¹ Since we remove more than z outliers, it is possible to have an approximation ratio $\alpha < 1$, i.e., $\phi_\epsilon(X, A) < r_{opt}$.

■ **Algorithm 1** Bi-criteria Approximation Algorithm.

Input: An instance (X, d) of metric k -center clustering with z outliers, and $|X| = n$; parameters $\epsilon > 0$, $\eta \in (0, 1)$, and $t \in \mathbb{Z}^+$.

1. Let $\gamma = z/n$ and initialize a set $E = \emptyset$.
2. Initially, $j = 1$; randomly select $\frac{1}{1-\gamma} \log \frac{1}{\eta}$ vertices from X and add them to E .
3. Run the following steps until $j = t$:
 - a. $j = j + 1$ and let Q_j be the farthest $(1 + \epsilon)z$ vertices of X to E (for each vertex $p \in X$, its distance to E is $\min_{q \in E} d(p, q)$).
 - b. Randomly select $\frac{1+\epsilon}{\epsilon} \log \frac{1}{\eta}$ vertices from Q_j and add them to E .

Output E .

2.1 $(2, O(\frac{1}{\epsilon}))$ -Approximation

Here, we consider bi-criteria approximation that returns more than k cluster centers. The main challenge for implementing Gonzalez's algorithm is that the outliers and inliers are mixed in X ; for example, the selected vertex, which has the largest minimum distance to the already selected vertices, is very likely to be an outlier, and therefore the clustering quality could be arbitrarily bad. Instead, our strategy is to take a small sample from the farthest subset. We implement our idea in Algorithm 1. For simplicity, let γ denote z/n in the algorithm; usually we can assume that γ is a value much smaller than 1. We prove the correctness of Algorithm 1 below.

► **Lemma 6.** *With probability at least $1 - \eta$, the set E in Step 2 of Algorithm 1 contains at least one point from X_{opt} .*

Since $|X_{opt}|/|X| = 1 - \gamma$, Lemma 6 can be easily obtained by the following folklore claim.

▷ **Claim 7.** Let U be a set of elements and $V \subseteq U$ with $\frac{|V|}{|U|} = \tau > 0$. Given $\eta \in (0, 1)$, if one randomly samples $\frac{1}{\tau} \log \frac{1}{\eta}$ elements from U , with probability at least $1 - \eta$, the sample contains at least one element from V .

Recall that $\{C_1, C_2, \dots, C_k\}$ are the k clusters forming X_{opt} . Denote by $\lambda_j(E)$ the number of the clusters which have non-empty intersection with E at the beginning of j -th round in Step 3 of Algorithm 1. For example, initially $\lambda_1(E) \geq 1$ by Lemma 6. Obviously, if $\lambda_j(E) = k$, i.e., $C_l \cap E \neq \emptyset$ for any $1 \leq l \leq k$, E yields a 2-approximation for k -center clustering with outliers through the triangle inequality.

▷ **Claim 8.** If $\lambda_j(E) = k$, then $\phi_0(X, E) \leq 2r_{opt}$.

► **Lemma 9.** *In each round of Step 3 of Algorithm 1, with probability at least $1 - \eta$, either (1) $d(Q_j, E) \leq 2r_{opt}$ or (2) $\lambda_j(E) \geq \lambda_{j-1}(E) + 1$.*

Proof. Suppose that (1) is not true, i.e., $d(Q_j, E) > 2r_{opt}$, and we prove that (2) is true. Let \mathcal{J} include all the indices $l \in \{1, 2, \dots, k\}$ with $E \cap C_l \neq \emptyset$. We claim that $Q_j \cap C_l = \emptyset$ for each $l \in \mathcal{J}$. Otherwise, let $p \in Q_j \cap C_l$ and $p' \in E \cap C_l$; due to the triangle inequality, we know that $d(p, p') \leq 2r_{opt}$ which is in contradiction to the assumption $d(Q_j, E) > 2r_{opt}$. Thus, $Q_j \cap X_{opt}$ only contains the vertices from C_l with $l \notin \mathcal{J}$. Moreover, since the number of outliers is z , we know that $\frac{|Q_j \cap X_{opt}|}{|Q_j|} \geq \frac{\epsilon}{1+\epsilon}$. By Claim 7, if randomly selecting $\frac{1+\epsilon}{\epsilon} \log \frac{1}{\eta}$ vertices from Q_j , with probability at least $1 - \eta$, the sample contains at least one vertex from $Q_j \cap X_{opt}$; also, the vertex must come from $\cup_{l \notin \mathcal{J}} C_l$. That is, (2) $\lambda_j(E) \geq \lambda_{j-1}(E) + 1$ happens. ◀

40:6 Greedy Strategy for k -Center Clustering with Outliers

If (1) of Lemma 9 happens, i.e., $d(Q_j, E) \leq 2r_{opt}$, then it implies that

$$\max_{p \in X \setminus Q_j} d(p, E) \leq 2r_{opt}; \quad (2)$$

moreover, since $|Q_j| = (1 + \epsilon)z$, we have $\phi_\epsilon(X, E) \leq 2r_{opt}$. Next, we assume that (1) in Lemma 9 never happens, and prove that $\lambda_j(E) = k$ with constant probability when $j = \Theta(k)$. The following idea actually has been used by Aggarwal et al. [1] for achieving a bi-criteria approximation for k -means clustering. Define a random variable x_j : $x_j = 1$ if $\lambda_j(E) = \lambda_{j-1}(E)$, or 0 if $\lambda_j(E) \geq \lambda_{j-1}(E) + 1$, for $j = 1, 2, \dots$. So $\mathbb{E}[x_j] \leq \eta$ by Lemma 9 and

$$\sum_{1 \leq s \leq j} (1 - x_s) \leq \lambda_j(E). \quad (3)$$

Also, let $J_j = \sum_{1 \leq s \leq j} (x_s - \eta)$ and $J_0 = 0$. Then, $\{J_0, J_1, J_2, \dots\}$ is a super-martingale with $J_{j+1} - J_j < 1$ (more details are shown in the full version of our paper). Through *Azuma-Hoeffding inequality* [4], we have $Pr(J_t \geq J_0 + \delta) \leq e^{-\frac{\delta^2}{2t}}$ for any $t \in \mathbb{Z}^+$ and $\delta > 0$. Let $t = \frac{k + \sqrt{k}}{1 - \eta}$ and $\delta = \sqrt{k}$, the inequality implies that

$$Pr\left(\sum_{1 \leq s \leq t} (1 - x_s) \geq k\right) \geq 1 - e^{-\frac{1-\eta}{4}}. \quad (4)$$

Combining (3) and (4), we know that $\lambda_t(E) \geq k$ with probability at least $1 - e^{-\frac{1-\eta}{4}}$. Moreover, $\lambda_t(E) = k$ directly implies that E is a 2-approximate solution by Claim 8. Together with Lemma 6, we have the following theorem.

► **Theorem 10.** *Let $\epsilon > 0$. If we set $t = \frac{k + \sqrt{k}}{1 - \eta}$ for Algorithm 1, with probability at least $(1 - \eta)(1 - e^{-\frac{1-\eta}{4}})$, $\phi_\epsilon(X, E) \leq 2r_{opt}$.*

Quality and Running time. If $\frac{1}{\eta}$ and $\frac{1}{1-\eta}$ are constant numbers, $|E|$ will be $O(\frac{k}{\epsilon})$ and Theorem 10 implies that E is a $(2, O(\frac{1}{\epsilon}))$ -approximation for $(k, z)_\epsilon$ -center clustering of X with constant probability. In each round of Step 3, there are $O(\frac{1}{\epsilon})$ new vertices added to E , thus it takes $O(\frac{1}{\epsilon}n)$ time to update the distances from the vertices of X to E ; to select the set Q_j , we can apply the linear time selection algorithm [10]. Overall, the running time of Algorithm 1 is $O(\frac{k}{\epsilon}n)$. If the given instance is in \mathbb{R}^D , the running time will be $O(\frac{k}{\epsilon}nD)$.

Further, we consider the instances under some practical assumption, and provide new analysis of Algorithm 1. In reality, the clusters are usually not too small, compared with the number of outliers. For example, it is rare to have a cluster C_l that $|C_l| \ll z$.

► **Theorem 11.** *If each optimal cluster C_l has size at least ϵz for $1 \leq l \leq k$, the set E of Algorithm 1 is a $(4, O(\frac{1}{\epsilon}))$ -approximation for the problem of $(k, z)_0$ -center clustering with constant probability.*

Compared with Theorem 10, Theorem 11 shows that we can exactly exclude z outliers (rather than $(1 + \epsilon)z$), though the approximation ratio with respect to the radius becomes 4.

Proof of Theorem 11. We take a more careful analysis on the proof of Lemma 9. If (1) never happens, eventually $\lambda_j(E)$ will reach k and thus $\phi_0(X, E) \leq 2r_{opt}$ (Claim 8). So we focus on the case that (1) happens before $\lambda_j(E)$ reaching k . Suppose at j -th round, $d(Q_j, E) \leq 2r_{opt}$ but $\lambda_j(E) < k$. We consider two cases (i) there exists some $l_0 \notin \mathcal{J}$ such that $C_{l_0} \subseteq Q_j$ and (ii) otherwise.

■ **Algorithm 2** 2-Approximation Algorithm.

Input: An instance (X, d) of metric k -center clustering with z outliers, and $|X| = n$; a parameter $\epsilon > 0$.

1. Initialize a set $E = \emptyset$.
2. Let $j = 1$; randomly select one vertex from X and add it to E .
3. Run the following steps until $j = k$:
 - a. $j = j + 1$ and let Q_j be the farthest $(1 + \epsilon)z$ vertices to E .
 - b. Randomly select one vertex from Q_j and add it to E .

Output E .

For (i), we have $C_{l_0} \subseteq Q_j$ for some $l_0 \notin \mathcal{J}$. Note that we assume $|C_{l_0}| \geq \epsilon z$, i.e., $\frac{|C_{l_0}|}{|Q_j|} \geq \frac{\epsilon}{1+\epsilon}$. Using the same manner in the proof of Lemma 9, we know that (2) $\lambda_j(E) \geq \lambda_{j-1}(E) + 1$ happens with probability $1 - \eta$. Thus, if (i) is always true, we can continue Step 3 and eventually $\lambda_j(E)$ will reach k , that is, a $(2, O(\frac{1}{\epsilon}))$ -approximation of $(k, z)_0$ -center clustering is obtained with constant probability.

For (ii), we have $C_l \setminus Q_j \neq \emptyset$ for all $l \notin \mathcal{J}$. Together with the assumption $d(Q_j, E) \leq 2r_{opt}$, we know that there exists $q_l \in C_l \setminus Q_j$ (for each $l \notin \mathcal{J}$) such that $d(q_l, E) \leq d(Q_j, E) \leq 2r_{opt}$. Consequently, we have that $\forall q \in C_l$,

$$d(q, E) \leq \|q - q_l\| + d(q_l, E) \leq 4r_{opt} \text{ (see the left of Figure. 1).} \quad (5)$$

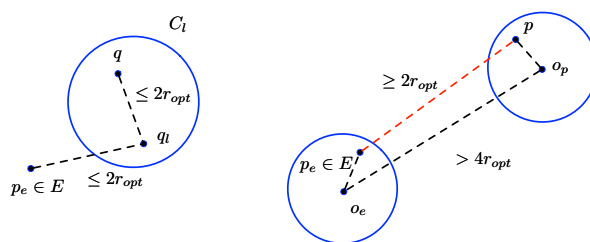
Note that for any $l \in \mathcal{J}$, $d(E, C_l) \leq 2r_{opt}$ by the triangle inequality. Thus,

$$\phi_0(X, E) \leq \max_{q \in \cup_{l=1}^k C_l} d(q, E) \leq 4r_{opt}. \quad (6)$$

So a $(4, O(\frac{1}{\epsilon}))$ -approximation of $(k, z)_0$ -center clustering is obtained. ◀

2.2 2-Approximation

If k is a constant, we show that a single-criterion 2-approximation can be achieved. Actually, we use the same strategy as Section 2.1, but run only k rounds with each round sampling only one vertex. See Algorithm 2.



■ **Figure 1** Left: p_e is a point of E having distance $\leq 2r_{opt}$ to p_l ; right: p_e is any point of E , o_e and o_p are the centers taking charge of p_e and p .

Denote by $\{v_1, \dots, v_k\}$ the k sampled vertices of E . Actually, the proof of Theorem 12 is similar to the analysis in Section 2.1. The only difference is that the probability that (2) $\lambda_j(E) \geq \lambda_{j-1}(E) + 1$ happens is at least $\frac{\epsilon}{1+\epsilon}$. Also note that $v_1 \in X_{opt}$ with probability $1 - \gamma$ ($\gamma = z/n$). If all of these events happen, either we obtain a 2-approximation before k steps

(i.e., $d(E, X \setminus Q_j) \leq 2r_{opt}$ for some $j < k$), or $\{v_1, \dots, v_k\}$ fall into the k optimal clusters C_1, C_2, \dots, C_k separately (i.e., $\lambda_k(E) = k$). No matter which case happens, we always obtain a 2-approximation with respect to $(k, z)_\epsilon$ -center clustering. So we have Theorem 12.

► **Theorem 12.** *Algorithm 2 returns a 2-approximation for the problem of $(k, z)_\epsilon$ -center clustering on X , with probability at least $(1 - \gamma)(\frac{\epsilon}{1+\epsilon})^{k-1}$. The running time is $O(kn)$. If the given instance is in \mathbb{R}^D , the running time will be $O(knD)$.*

To boost the probability of Theorem 12, we just need to repeatedly run the algorithm; the success probability is easy to calculate by taking the union bound.

► **Corollary 13.** *If we run Algorithm 2 $O(\frac{1}{1-\gamma}(\frac{1+\epsilon}{\epsilon})^{k-1})$ times, with constant probability, at least one time the algorithm returns a 2-approximation for the problem of $(k, z)_\epsilon$ -center clustering.*

Similar to Theorem 11, we consider the practical instances. We show that the quality of Theorem 12 can be preserved even exactly excluding z outliers, if the optimal clusters are “well separated”. The property was also studied for other clustering problems in practice [17, 26]. Let $\{o_1, \dots, o_k\}$ be the k cluster centers of the optimal clusters $\{C_1, \dots, C_k\}$.

► **Theorem 14.** *Suppose that each optimal cluster C_l has size at least ϵz and $\|o_l - o_{l'}\| > 4r_{opt}$ for $1 \leq l \neq l' \leq k$. Then with probability at least $(1 - \gamma)(\frac{\epsilon}{1+\epsilon})^{k-1}$, Algorithm 2 returns a 2-approximation for the problem of $(k, z)_0$ -center clustering.*

Proof. Initially, we know that $\lambda_1(E) = 1$ with probability $1 - \gamma$. Suppose that at the beginning of the j -th round of Algorithm 2 with $2 \leq j \leq k$, E already has $j - 1$ vertices separately falling in $j - 1$ optimal clusters; also, we still let \mathcal{J} be the set of the indices of these $j - 1$ clusters. Then we have the following claim.

▷ **Claim 15.** $|Q_j \cap (\cup_{l \notin \mathcal{J}} C_l)| \geq \epsilon z$.

Proof. For any $p \in \cup_{l \notin \mathcal{J}} C_l$, we have

$$d(p, E) > 4r_{opt} - r_{opt} - r_{opt} = 2r_{opt} \quad (7)$$

from triangle inequality and the assumption $\|o_l - o_{l'}\| > 4r_{opt}$ for $1 \leq l \neq l' \leq k$ (see the right of Figure. 1). In addition, for any $p \in \cup_{l \in \mathcal{J}} C_l$, we have

$$d(p, E) \leq 2r_{opt}. \quad (8)$$

We consider two cases. If $d(Q_j, E) \leq 2r_{opt}$ at the current round, then (7) directly implies that $\cup_{l \notin \mathcal{J}} C_l \subseteq Q_j$ (recall Q_j is the set of farthest vertices to E); thus $|Q_j \cap (\cup_{l \notin \mathcal{J}} C_l)| = |\cup_{l \notin \mathcal{J}} C_l| \geq \epsilon z$ by the assumption that any $|C_l| \geq \epsilon z$. Otherwise, $d(Q_j, E) > 2r_{opt}$. Then $Q_j \cap (\cup_{l \in \mathcal{J}} C_l) = \emptyset$ by (8). Moreover, since there are only z outliers and $|Q_j| = (1 + \epsilon)z$, we know that $|Q_j \cap (\cup_{l \notin \mathcal{J}} C_l)| \geq \epsilon z$. ◁

Claim 15 reveals that with probability at least $\frac{\epsilon}{1+\epsilon}$, the new added vertex falls in $\cup_{l \notin \mathcal{J}} C_l$, i.e., $\lambda_j(E) = \lambda_{j-1}(E) + 1$. Overall, we know that $\lambda_k(E) = k$, i.e., E is a 2-approximation of $(k, z)_0$ -center clustering (by Claim 8), with probability at least $(1 - \gamma)(\frac{\epsilon}{1+\epsilon})^{k-1}$. ◀

2.3 Reducing Data Size via Random Sampling

Given a metric (X, d) , Charikar et al. [16] showed that we can use a random sample S to replace X . Recall $\gamma = z/n$. Let $|S| = O(\frac{k}{\epsilon^2 \gamma} \ln n)$ and E be an α -approximate solution of $(k, z)_\epsilon$ -center clustering on (S, d) , then E is an α -approximate solution of $(k, z)_{O(\epsilon)}$ -center clustering on (X, d) with constant probability. In D -dimensional Euclidean space, Huang et al. [23] showed a similar result, where the sample size $|S| = \tilde{O}(\frac{1}{\epsilon^2 \gamma^2} kD)^2$ (to be consistent with our paper, we change the notations in their theorem). In this section, we show that the sample size of [23] can be further improved to be $\tilde{O}(\frac{1}{\epsilon^2 \gamma} kD)$, which can be a significant improvement if $\frac{1}{\gamma} = \frac{n}{z}$ is large.

Let P be a set of n points in \mathbb{R}^D . Consider the range space $\Sigma = (P, \Pi)$ where each range $\pi \in \Pi$ is the complement of union of k balls in \mathbb{R}^D . We know that the VC dimension of balls is $O(D)$ [4], and therefore the VC dimension of union of k balls is $O(kD \log k)$ [11]. That is, the VC dimension of the range space Σ is $O(kD \log k)$. Let $\epsilon \in (0, 1)$, and an “ ϵ -sample” S of P is defined as follows: $\forall \pi \in \Pi$, $|\frac{|\pi \cap P|}{|P|} - \frac{|\pi \cap S|}{|S|}| \leq \epsilon$; roughly speaking, S is an approximation of P with an additive error inside each range π . Given a range space with VC dimension m , an ϵ -sample can be easily obtained via uniform sampling [4], where the success probability is $1 - \lambda$ and the sample size is $O(\frac{1}{\epsilon^2} (m \log \frac{m}{\epsilon} + \log \frac{1}{\lambda}))$ for any $0 < \lambda < 1$. For our problem, we need to replace the “ ϵ ” of the “ ϵ -sample” by $\epsilon\gamma$ to guarantee that the number of uncovered points is bounded by $(1 + O(\epsilon))\gamma n$ (we show the details below); the resulting sample size will be $\tilde{O}(\frac{1}{\epsilon^2 \gamma^2} kD)$ that is the same as the sample size of [23] (we assume that the term $\log \frac{1}{\lambda}$ is a constant for convenience).

Actually, the front factor $\frac{1}{\epsilon^2 \gamma^2}$ of the sample size can be further reduced to be $\frac{1}{\epsilon^2 \gamma}$ by a more careful analysis. We observe that there is no need to guarantee the additive error for each range π (as the definition of ϵ -sample). Instead, only a multiplicative error for the ranges covering at least γn points should be sufficient. Note that when a range covers more points, the multiplicative error is weaker than the additive error and thus the sample size is reduced. For this purpose, we use *relative approximation* [21, 31]: let $S \subseteq P$ be a subset of size $\tilde{O}(\frac{1}{\epsilon^2 \gamma} kD)$ chosen uniformly at random, then with constant probability,

$$\forall \pi \in \Pi, \left| \frac{|\pi \cap P|}{|P|} - \frac{|\pi \cap S|}{|S|} \right| \leq \epsilon \times \max \left\{ \frac{|\pi \cap P|}{|P|}, \gamma \right\}. \quad (9)$$

We formally state our result below.

► **Theorem 16.** *Let P be an instance for the problem of k -center clustering with outliers in \mathbb{R}^D as described in Definition 1, and $S \subseteq P$ be a subset of size $\tilde{O}(\frac{1}{\epsilon^2 \gamma} kD)$ chosen uniformly at random. Suppose $\epsilon \leq 0.5$. Let S be a new instance for the problem of k -center clustering with outliers where the number of outliers is set to be $z' = (1 + \epsilon)\gamma|S|$. If E is an α -approximate solution of $(k, z')_\epsilon$ -center clustering on S , then E is an α -approximate solution of $(k, z)_{O(\epsilon)}$ -center clustering on P , with constant probability.*

Proof. We assume that S is a relative approximation of P and (9) holds (this happens with constant probability). Let \mathbb{B}_{opt} be the set of k balls covering $(1 - \gamma)n$ points induced by the optimal solution for P , and \mathbb{B}_S be the set of k balls induced by an α -approximate solution of $(k, z')_\epsilon$ -center clustering on S . Suppose the radius of each ball in \mathbb{B}_{opt} (resp., \mathbb{B}_S) is r_{opt} (resp., r_S). We denote the complements of \mathbb{B}_{opt} and \mathbb{B}_S as π_{opt} and π_S , respectively.

² The asymptotic notation $\tilde{O}(f) = O(f \cdot \text{polylog}(\frac{kD}{\epsilon\gamma}))$.

40:10 Greedy Strategy for k -Center Clustering with Outliers

First, since \mathbb{B}_{opt} covers $(1 - \gamma)n$ points of P and S is a relative approximation of P , we have

$$\frac{|\pi_{opt} \cap S|}{|S|} \leq \frac{|\pi_{opt} \cap P|}{|P|} + \epsilon \times \max \left\{ \frac{|\pi_{opt} \cap P|}{|P|}, \gamma \right\} = (1 + \epsilon)\gamma \quad (10)$$

by (9). That is, the set balls \mathbb{B}_{opt} cover at least $(1 - (1 + \epsilon)\gamma)|S|$ points of S , and therefore it is a feasible solution for the instance S with respect to the problem of k -center clustering with z' outliers. Since \mathbb{B}_S is an α -approximate solution of (k, z') -center clustering on S , we have

$$r_S \leq \alpha r_{opt}; \quad |\pi_S \cap S| \leq (1 + \epsilon)z' = (1 + \epsilon)^2 \gamma |S|. \quad (11)$$

Now, we claim that

$$|\pi_S \cap P| \leq \frac{(1 + \epsilon)^2}{1 - \epsilon} \gamma |P|. \quad (12)$$

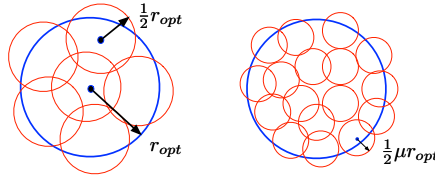
Assume that (12) is not true, then (9) implies $\left| \frac{|\pi_S \cap P|}{|P|} - \frac{|\pi_S \cap S|}{|S|} \right| \leq \epsilon \times \max \left\{ \frac{|\pi_S \cap P|}{|P|}, \gamma \right\} = \epsilon \frac{|\pi_S \cap P|}{|P|}$. So $\frac{|\pi_S \cap S|}{|S|} \geq (1 - \epsilon) \frac{|\pi_S \cap P|}{|P|} > (1 + \epsilon)^2 \gamma$, which is in contradiction with the second inequality of (11), and thus (12) is true. We assume $\epsilon \leq 0.5$, so $\frac{1}{1 - \epsilon} \leq 1 + 2\epsilon$ and $\frac{(1 + \epsilon)^2}{1 - \epsilon} = 1 + O(\epsilon)$. Consequently (12) and the first inequality of (11) together imply that \mathbb{B}_S is an α -approximate solution of $(k, z)_{O(\epsilon)}$ -center clustering on P . ◀

3 Coreset Construction in Doubling Metrics

In this section, we always assume the following is true by default:

Given an instance (X, d) of k -center clustering with outliers, the metric (X_{opt}, d) , i.e., the metric formed by the set of inliers, has a constant doubling dimension $\rho > 0$.

We do not have any restriction on the outliers $X \setminus X_{opt}$. Thus the above assumption is more relaxed and practical than assuming the whole (X, d) has a constant doubling dimension. From Definition 5, we directly know that each optimal cluster C_l of X_{opt} can be covered by 2^ρ balls with radius $r_{opt}/2$ (see the left figure in Figure. 2). Imagine that the instance (X, d) has $2^\rho k$ clusters, where the optimal radius is at most $r_{opt}/2$. Therefore, we can just replace k by $2^\rho k$ when running Algorithm 1, so as to reduce the approximation ratio (i.e., the ratio of the resulting radius to r_{opt}) from 2 to 1.



■ **Figure 2** Illustrations for Theorem 17 and 18.

► **Theorem 17.** *If we set $t = \frac{2^\rho k + 2^{\rho/2} \sqrt{k}}{1 - \eta}$ for Algorithm 1, with probability $(1 - \eta)(1 - e^{-\frac{1 - \eta}{4}})$, $\phi_\epsilon(X, E) \leq r_{opt}$. So the set E is a $(1, O(\frac{2^\rho}{\epsilon}))$ -approximation for the problem of $(k, z)_\epsilon$ -center clustering, and the running time is $O(2^\rho \frac{k}{\epsilon} n)$.*

■ **Algorithm 3** The Coreset Construction.

Input: An instance (X, d) of metric k -center clustering with z outliers, and $|X| = n$; parameters η and $\mu \in (0, 1)$.

1. Let $l = (\frac{2}{\mu})^\rho k$.
2. Set $\epsilon = 1$ and run Algorithm 1 $t = \frac{l + \sqrt{l}}{1 - \eta}$ rounds. Denote by \tilde{r} the maximum distance between E and X by excluding the farthest $2z$ vertices, after the final round of Algorithm 1.
3. Let $X_{\tilde{r}} = \{p \mid p \in X \text{ and } d(p, E) \leq \tilde{r}\}$.
4. For each vertex $p \in X_{\tilde{r}}$, assign it to its nearest neighbor in E ; for each vertex $q \in E$, let its weight be the number of vertices assigning to it.
5. Add $X \setminus X_{\tilde{r}}$ to E ; each vertex of $X \setminus X_{\tilde{r}}$ has weight 1.

Output E as the coreset.

If considering the problem in Euclidean space \mathbb{R}^D where the doubling dimension of the inliers is ρ , the running time becomes $O(2^\rho \frac{k}{\epsilon} n D)$. Inspired by Theorem 17, we can further construct coreset for k -center clustering with outliers (see Definition 3). Let $\mu \in (0, 1)$, and for simplicity we assume that $\log 2/\mu$ is an integer. If applying Definition 5 recursively, we know that each C_l is covered by $2^{\rho \log 2/\mu} = (\frac{2}{\mu})^\rho$ balls with radius $\frac{\mu}{2} r_{opt}$, and X_{opt} is covered by $(\frac{2}{\mu})^\rho k$ such balls in total. See the right figure in Figure. 2. Based on this observation, we have Algorithm 3 for constructing μ -coreset.

► **Theorem 18.** *With constant probability, Algorithm 3 outputs a μ -coreset E of k -center clustering with z outliers. The size of E is at most $2z + O((\frac{2}{\mu})^\rho k)$, and the construction time is $O((\frac{2}{\mu})^\rho kn)$.*

Remark. (1) The previous ideas based on uniform sampling [16, 23] (also our idea in Section 2.3) cannot get rid of the violation on the number of outliers; the sample sizes will become infinity if not allowing to remove more than z outliers. Our coreset in Theorem 18 works for removing z outliers exactly. Consequently, our coreset can be used for existing algorithms of k -center clustering with outliers, such as [15], to reduce their complexities. (2) Another feature is that our coreset is a natural composable coreset. If X (or the point set P) is partitioned into L parts, we can run Algorithm 3 for each part, and obtain a coreset with size $(2z + O((\frac{2}{\mu})^\rho k))L$ in total (the proof is almost identical to the proof of Theorem 18 below). So our coreset construction can potentially be applied to distributed clustering with outliers. (3) The coefficient 2 of z actually can be further reduced by modifying the value of ϵ in Step 2 of Algorithm 3 (we just set $\epsilon = 1$ for simplicity). In general, the size of E is $(1 + \epsilon)z + O(\frac{1}{\epsilon}(\frac{2}{\mu})^\rho k)$ and the construction time is $O(\frac{1}{\epsilon}(\frac{2}{\mu})^\rho kn)$ (or $O(\frac{1}{\epsilon}(\frac{2}{\mu})^\rho kn D)$ in \mathbb{R}^D).

Proof of Theorem 18. Similar to Theorem 17, we know that $|X_{\tilde{r}}| = n - 2z$ and $\tilde{r} \leq 2 \times \frac{\mu}{2} r_{opt} = \mu r_{opt}$ with constant probability in Algorithm 3. Thus, the size of E is $|X \setminus X_{\tilde{r}}| + O((\frac{2}{\mu})^\rho k) = 2z + O((\frac{2}{\mu})^\rho k)$. Moreover, it is easy to see that the running time of Algorithm 3 is $O((\frac{2}{\mu})^\rho kn)$.

Next, we show that E is a μ -coreset of X . For each vertex $q \in E$, denote by $w(q)$ the weight of q ; for the sake of convenience in our proof, we view each q as a set of $w(q)$ overlapping unit weight vertices. Thus, from the construction of E , we can see that there is a bijective mapping f between X and E , where

$$\|p - f(p)\| \leq \tilde{r} \leq \mu r_{opt}, \quad \forall p \in X. \quad (13)$$

40:12 Greedy Strategy for k -Center Clustering with Outliers

Let $H = \{c_1, c_2, \dots, c_k\}$ be any k vertices of X . Suppose that H induces k clusters $\{A_1, A_2, \dots, A_k\}$ (resp., $\{B_1, B_2, \dots, B_k\}$) with respect to the problem of k -center clustering with z outliers on E (resp., X), where each A_j (resp., B_j) has the cluster center c_j for $1 \leq j \leq k$. Let $r_E = \phi_0(E, H)$ and $r_X = \phi_0(X, H)$, respectively. Also, let r'_E (resp., r'_X) be the smallest value r , such that for any $1 \leq j \leq k$, $f(B_j) \subseteq \text{Ball}(c_j, r)$ (resp., $f^{-1}(A_j) \subseteq \text{Ball}(c_j, r)$). We need the following claim.

▷ **Claim 19.** $|r'_E - r_X| \leq \mu r_{opt}$ and $|r'_X - r_E| \leq \mu r_{opt}$.

In addition, since $\{f(B_1), \dots, f(B_k)\}$ also form k clusters for the instance E with the fixed k cluster centers of H , we know that $r'_E \geq \phi_0(E, H) = r_E$. Similarly, we have $r'_X \geq r_X$. Combining Claim 19, we have

$$r_X - \mu r_{opt} \leq \underbrace{r'_X - \mu r_{opt}}_{\text{by Claim 19}} \leq r_E \leq \underbrace{r'_E}_{\text{by Claim 19}} \leq r_X + \mu r_{opt}. \quad (14)$$

So $|r_X - r_E| \leq \mu r_{opt}$, i.e., $\phi_0(E, H) \in \phi_0(X, H) \pm \mu r_{opt} \subseteq (1 \pm \mu)\phi_0(X, H)$. Therefore E is a μ -coreset of (X, d) . ◀

4 Experiments

Our experimental results were obtained on a Windows workstation with 2.8GHz Intel(R) Core(TM) i5-840 and 8GB main memory; the algorithms were implemented in Matlab R2018a. We test our algorithms on both synthetic and real datasets. For Algorithm 2, we take two well known algorithms of k -center clustering with outliers, *Base1* of [15] and *Base2* of [33], as the baselines. For Algorithm 3, we compare our coreset construction with uniform random sampling.

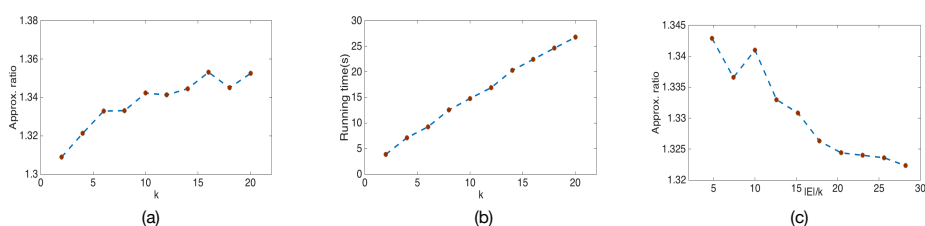
To generate the synthetic datasets, we set $n = 10^5$ and $D = 10^3$, and vary the values of z and k . First, randomly generate k clusters inside a hypercube of side length 200, where each cluster is a random sample from a Gaussian distribution with variance 10; each cluster has a random number of points and we keep the total number of points to be $n - z$; we compute the minimum enclosing balls respectively for these k clusters (by using the algorithm of [6]), and randomly generate z outliers outside the balls. The maximum radius of the balls is used as r_{opt} .

We also use three real datasets. MNIST dataset [28] contains $n = 60,000$ handwritten digit images from 0 to 9, where each image is represented by a 784-dimensional vector. The 10 digits form $k = 10$ clusters. Caltech-256 dataset [29] contains 30,607 colored images with 256 categories, where each image is represented by a 4096-dimensional vector. We choose $n = 2,232$ images of 20 categories to form $k = 20$ clusters. CIFER-10 training dataset [27] contains $n = 50,000$ colored images in 10 classes as $k = 10$ clusters, where each image is represented by a 4096-dimensional vector. For each real dataset, we use the minimum enclosing ball algorithm of [6] to compute r_{opt} , and randomly generate $z = 5\%n$ outliers outside the corresponding balls.

Results and analysis. Note that we exactly exclude z outliers (rather than $(1 + \epsilon)z$ as stated in Theorem 10 and 12) in our experiments, and calculate the approximation ratio $\phi_0(X, E)/r_{opt}$ for each instance, if E is the set of returned cluster centers.

We first run our Algorithm 1 on synthetic and real datasets. For synthetic datasets, we set $k = 2-20$, and $\beta = |E|/k = 8$ via modifying the values of ϵ and η appropriately (that means we output $8k$ cluster centers); normally, we set $\eta = 0.1$ and $\epsilon \approx 0.7$. We try the instances with

$z = \{2\%n, 4\%n, 6\%n, 8\%n, 10\%n\}$, and report the average results in Figure 3 (a) and (b); the approximation ratios are within 1.3-1.4 and the running times are less than 30s. Actually, the performance is quite stable regarding different values of z in our experiments, and the standard variances of approximation ratios and running times are less than 0.03 and 0.12, respectively. We also vary the value of β from 4 to 28 with $k = 10$. Figure 3 (c) shows that the approximation ratio slightly decreases as β increases. The running times are all around 14s and do not reveal a clear increasing trend as β increases. We think the reason behind may be that we just use the simple $O(n \log n)$ sorting algorithm, rather than the linear time selection algorithm [10], for computing Q_j in practice (see Step 3(a) of Algorithm 1); thus the running time is not linearly dependent on $|E|$. The results for real datasets are shown in the full version of our paper; the approximation ratios are all below 1.3 and the running times are less than 35s even for the largest CIFER-10 dataset.



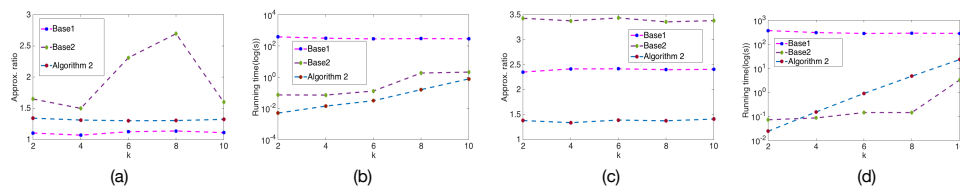
■ **Figure 3** The experimental results of Algorithm 1 on synthetic datasets.

We also test our Algorithm 2 on synthetic and real datasets. We set $\epsilon = 1$ so that to avoid to repeat running Algorithm 2 too many times (see Corollary 13), but we still exactly exclude z outliers for calculating the approximation ratio as mentioned before. Our results are shown in Table 1. The synthetic and real datasets are too large for the baseline algorithms *Base1* and *Base2*, e.g., they run too slowly or even out of memory in our workstation if n , z , and D are large (they have complexities $\Omega(n^2D)$ or $\Omega(kznD)$)³. To make a fair comparison, we run *Base1*, *Base2*, and Algorithm 2 on smaller synthetic datasets with $(n = 2000, D = 10)$ and $(n = 2000, D = 100)$; we also set $z = \{2\%n, 4\%n, 6\%n, 8\%n, 10\%n\}$ as before and report the average results. When $D = 10$, *Base1* and Algorithm 2 achieve approximation ratios < 1.5 generally (Figure 4 (a)); moreover, *Base2* and Algorithm 2 run much faster than *Base1* (Figure 4 (b)). However, when $D = 100$, *Base1* and *Base2* yield much worse approximation ratios than Algorithm 2 (Figure 4 (c) and (d)). Our experiment reveals that Algorithm 2 can achieve a more stable performance when dimensionality increases.

Finally, we compare the performances of our coresets method (Algorithm 3) and uniform random sampling in terms of reducing data sizes. Though real-world image datasets often are believed to have low intrinsic dimensions [8], it is difficult to compute them (e.g., doubling dimension) accurately. In practice, we can directly set an appropriate value for l in Step 1 of Algorithm 3 (without knowing the value of doubling dimension ρ). For example, the size of coreset is $2z + O(\left(\frac{2}{\mu}\right)^\rho k) = 2z + O(l)$ according to Theorem 18, so we keep the sizes of our coresets to be $\{15\%n, 20\%n, 25\%n\}$ via modifying the value of l in our experiments. Correspondingly, we also set the sizes of random samples to be $\{15\%n, 20\%n, 25\%n\}$. We run Algorithm 2 on the corresponding random samples and coresets, and report the results in Table 2. Running Algorithm 2 on the coresets yields approximation ratios close to

³ We are aware of several distributed algorithms for k -center clustering with outliers [12, 19, 30, 32], but we only consider the setting with single machine in this paper.

40:14 Greedy Strategy for k -Center Clustering with Outliers



■ **Figure 4** Comparison of Base1, Base2, and Algorithm 2 on smaller synthetic datasets ((a) and (b) for $D = 10$; (c) and (d) for $D = 100$).

■ **Table 1** The results of Algorithm 2 on synthetic and real datasets.

	Synthetic datasets				Real datasets		
	k=2	k=4	k=6	k=8	MNIST	CALTECH256	CIFAR10
Approx. ratio	1.410	1.403	1.406	1.423	1.277	1.378	1.249
Running time(s)	8.097	63.636	374.057	1939.004	2644.709	2864.231	13295.306

those obtained by directly running the algorithm on the original datasets; the results also remain stably when the level reduces from 25% to 15%. More importantly, our coresets significantly reduce the running times (e.g., it only needs 15%-35% time by using 15%-level coreset). Comparing with the random samples, our coresets can achieve significantly lower approximation ratios especially for the 15% level. Note that the coreset based approach takes more time than uniform random sampling, because we count the time spent for coreset construction.

■ **Table 2** The results of Algorithm 2 on random samples, coresets, and original datasets.

		random sampling			coreset			100%
		15%	20%	25%	15%	20%	25%	
MNIST	Appro. Ratio	1.591	1.597	1.566	1.275	1.261	1.261	1.277
	running time(s)	624.612	769.517	958.549	936.393	1071.926	1262.996	2644.709
CALTECH256	Appro. Ratio	2.144	1.779	1.448	1.431	1.420	1.401	1.378
	running time(s)	407.294	510.423	603.713	413.961	516.862	609.979	2864.231
CIFAR10	Appro. Ratio	1.538	1.383	1.446	1.248	1.256	1.249	1.249
	running time(s)	2420.943	2170.416	2938.773	3526.752	3264.858	4033.862	13295.306

5 Future Work

Following our work, several interesting problems deserve to be studied in future. For example, can the coreset construction time of Algorithm 3 be improved, like the fast net construction method proposed by Har-Peled and Mendel [20] in doubling metrics? It is also interesting to study other problems involving outliers by using greedy strategy.

References

- 1 Ankit Aggarwal, Amit Deshpande, and Ravi Kannan. Adaptive sampling for k-means clustering. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 15–28. Springer, 2009.
- 2 Sepideh Aghamolaei and Mohammad Ghodsi. A Composable Coreset for k-Center in Doubling Metrics. In *Proceedings of the 30th Canadian Conference on Computational Geometry, CCCG 2018, August 8-10, 2018, University of Manitoba, Winnipeg, Manitoba, Canada*, pages 165–171, 2018.

- 3 Noga Alon, Seannie Dar, Michal Parnas, and Dana Ron. Testing of clustering. *SIAM Journal on Discrete Mathematics*, 16(3):393–417, 2003.
- 4 Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2004.
- 5 Olivier Bachem, Mario Lucic, and Andreas Krause. Practical coresets constructions for machine learning. *arXiv preprint arXiv:1703.06476*, 2017. [arXiv:1703.06476](#).
- 6 Mihai Badoiu and Kenneth L. Clarkson. Smaller core-sets for balls. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 801–802, 2003.
- 7 Mihai Badoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 250–257, 2002.
- 8 Mikhail Belkin. *Problems of learning on manifolds*. The University of Chicago, 2003.
- 9 Avrim Blum, John Hopcroft, and Ravindran Kannan. Foundations of data science. *Vorabversion eines Lehrbuchs*, 2016.
- 10 Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.
- 11 Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989.
- 12 Matteo Ceccarello, Andrea Pietracaprina, and Geppino Pucci. Solving k-center Clustering (with Outliers) in MapReduce and Streaming, almost as Accurately as Sequentially. *CoRR*, abs/1802.09205, 2018.
- 13 Deeparnab Chakrabarty, Prachi Goyal, and Ravishankar Krishnaswamy. The Non-Uniform k-Center Problem. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 67:1–67:15, 2016.
- 14 Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.
- 15 Moses Charikar, Samir Khuller, David M Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 642–651. Society for Industrial and Applied Mathematics, 2001.
- 16 Moses Charikar, Liadan O’Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 30–39. ACM, 2003.
- 17 Amit Daniely, Nati Linial, and Michael E. Saks. Clustering is difficult only when it does not matter. *CoRR*, abs/1205.4891, 2012.
- 18 Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
- 19 Sudipto Guha, Yi Li, and Qin Zhang. Distributed Partial Clustering. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 143–152. ACM, 2017.
- 20 Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, 2006.
- 21 Sariel Har-Peled and Micha Sharir. Relative (p, ϵ) -approximations in geometry. *Discrete & Computational Geometry*, 45(3):462–496, 2011.
- 22 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.
- 23 Lingxiao Huang, Shaofeng Jiang, Jian Li, and Xuan Wu. Epsilon-Coresets for Clustering (with Outliers) in Doubling Metrics. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 814–825. IEEE, 2018.
- 24 Piotr Indyk, Sepideh Mahabadi, Mohammad Mahdian, and Vahab S. Mirrokni. Composable core-sets for diversity and coverage maximization. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS’14, Snowbird, UT, USA, June 22-27, 2014*, pages 100–108, 2014.
- 25 Anil K Jain. Data clustering: 50 years beyond K-means. *Pattern recognition letters*, 31(8):651–666, 2010.

40:16 Greedy Strategy for k -Center Clustering with Outliers

- 26 Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. An Efficient k -Means Clustering Algorithm: Analysis and Implementation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):881–892, 2002.
- 27 Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.
- 28 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.
- 29 Fei-Fei Li, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer vision and Image understanding*, 106(1):59–70, 2007.
- 30 Shi Li and Xiangyu Guo. Distributed k -Clustering for Data with Heavy Noise. In *Advances in Neural Information Processing Systems*, pages 7849–7857, 2018.
- 31 Yi Li, Philip M Long, and Aravind Srinivasan. Improved bounds on the sample complexity of learning. *Journal of Computer and System Sciences*, 62(3):516–527, 2001.
- 32 Gustavo Malkomes, Matt J Kusner, Wenlin Chen, Kilian Q Weinberger, and Benjamin Moseley. Fast distributed k -center clustering with outliers on massive data. In *Advances in Neural Information Processing Systems*, pages 1063–1071, 2015.
- 33 Richard Matthew McCutchen and Samir Khuller. Streaming Algorithms for k -Center Clustering with Outliers and with Anonymity. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 165–178. Springer, 2008.
- 34 Jeff M. Phillips. Coresets and Sketches. *Computing Research Repository*, 2016.
- 35 Kunal Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 281–290, 2004.

Bidirectional Text Compression in External Memory

Patrick Dinklage

Technische Universität Dortmund, Department of Computer Science, Germany
patrick.dinklage@tu-dortmund.de

Jonas Ellert

Technische Universität Dortmund, Department of Computer Science, Germany
jonas.ellert@tu-dortmund.de

Johannes Fischer

Technische Universität Dortmund, Department of Computer Science, Germany
johannes.fischer@cs.tu-dortmund.de

Dominik Köppl 

Kyushu University, Fukuoka, Japan Society for Promotion of Science, Japan
<https://dkppl.de/>
dominik.koeppel@inf.kyushu-u.ac.jp

Manuel Penschuck

Goethe University Frankfurt, Department of Computer Science, Germany
mpenschuck@ae.cs.uni-frankfurt.de

Abstract

Bidirectional compression algorithms work by substituting repeated substrings by references that, unlike in the famous LZ77-scheme, can point to either direction. We present such an algorithm that is particularly suited for an external memory implementation. We evaluate it experimentally on large data sets of size up to 128 GiB (using only 16 GiB of RAM) and show that it is significantly faster than all known LZ77 compressors, while producing a roughly similar number of factors. We also introduce an external memory decompressor for texts compressed with any uni- or bidirectional compression scheme.

2012 ACM Subject Classification Theory of computation → Data compression

Keywords and phrases text compression, bidirectional parsing, text decompression, external algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.41

Related Version A full version of the paper is available at <https://arxiv.org/abs/1907.03235>.

Funding *Dominik Köppl*: JSPS KAKENHI Grant Number JP18F18120

Manuel Penschuck: Deutsche Forschungsgemeinschaft (DFG) grants ME 2088/3-2, ME 2088/4-2

1 Introduction

Text compression is a fundamental task when storing massive data sets. Most practical text compressors such as gzip, bzip2, 7zip, etc., scan a text file with a sliding window, replacing repetitive occurrences within this window. Although this approach is memory and time efficient [3, 29], two occurrences of the same substring are neglected if their distance is longer than the sliding window. More advanced solutions [12, 13, 9, 19, to mention only a few examples] drop the idea of a sliding window, thereby finding also repetitions that are far apart in the text. These so-called LZ77-algorithms have a better compression ratio in practice [8, Sect. 6]. In recent years, these algorithms have also been transformed to the *external memory* (EM) model [17, 21, 2].



© Patrick Dinklage, Jonas Ellert, Johannes Fischer, Dominik Köppl, and Manuel Penschuck; licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 41; pp. 41:1–41:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this article, we present a modification of *LZ77*, called *plcpcomp*, which is based on the bidirectional compression scheme *lcpcomp* of [6], but is better suited for an efficient external memory implementation due to its memory access patterns. We can compute this scheme by scanning the text and two auxiliary arrays stored in EM (one of them being the *permuted longest common prefix array*, hence the acronym *plcp*). We underline the performance of our algorithm with evaluations showing that it is faster than any known *LZ77* compressor for massive non-highly repetitive data sets. We also present the first external decompressor for files that are compressed with a bidirectional scheme.

1.1 Related Work

Our work is the first to join the fields of bidirectional and external memory compression.

1.1.1 Bidirectional Schemes

First considerations started with [29] who also coined this notation. [11] proved that finding the *optimal* bidirectional parsing, i.e., a bidirectional parsing with the lowest number of factors, is NP-complete. [6] were the first to present a greedy algorithm for producing a bidirectional parsing called *lcpcomp*, which performs well in practice, but comes with no theoretical performance guarantees on its size. [25] combined the techniques for *lcpcomp* [6] and the longest-first grammar compression [26] in a compression algorithm running in $\mathcal{O}(n^2)$ time, which was subsequently improved to $\mathcal{O}(n \lg n)$ time by [27]. Recently, [10] showed an upper bound of $z = \mathcal{O}(b \lg(n/b))$ and a lower bound of $z = \Omega(b \lg n)$ for some specific strings, where b and z denote the minimal number of factors in an optimal *bidirectional* parsing and in an optimal *unidirectional* parsing, respectively. This implies that bidirectional parsing can be exponentially better than unidirectional parsing. They also proposed a bidirectional parsing based on the Burrows-Wheeler transform (BWT). [22] introduced so-called *string attractors*, showed that a bidirectional scheme is a string attractor and that every string attractor can be represented with a bidirectional scheme. Last but not least, the bidirectional scheme of [28] guarantees to produce at most as many factors as *LZ77*, but has the disadvantage of a super-quadratic running time.

1.1.2 EM Compression Algorithms

Yanovsky [30] presented a compressor called *ReCoil* that is specialized on large DNA datasets. Ferragina et al. [7] gave a construction algorithm of the Burrows-Wheeler transform in EM. For *LZ77* compression, [17] devised two algorithms called *EM-LZscan* and *EM-LPF*. The former performs well on highly-repetitive data, but gets outperformed easily by *EM-LPF* on other kinds of datasets. The *LZ77* compressed files can be decompressed with an algorithm due to [2], which also works in general for all files that have been compressed by a unidirectional scheme. Finally, [21] presented an EM algorithm for computing the *LZ-End* scheme [23], a variant of *LZ77*.

1.2 Preliminaries

Model of computation. We use the commonly accepted EM model by Aggarwal and Vitter [1]. It features two memory types, namely fast internal memory (IM) which may hold up to M data words, and slow EM of unbounded size. The measure of the performance of an algorithm is the number of input and output operations (I/Os) required, where each I/O transfers a block of B consecutive words between memory levels. Reading or writing

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	a	b	a	b	b	a	b	a	b	a	b	b	a	b	b	a	a	b	a	b	a	\$
SA	22	21	16	19	17	6	1	8	13	3	10	20	15	18	5	7	12	2	9	14	4	11
ISA	7	18	10	21	15	6	16	8	19	11	22	17	9	20	13	3	5	14	4	12	2	1
Φ	6	12	13	14	18	17	5	1	2	3	4	7	8	9	20	21	19	15	16	10	22	11
LCP	0	0	1	1	3	5	4	7	2	4	5	0	2	2	4	5	3	5	6	1	3	4
$PLCP$	4	5	4	3	4	5	5	7	6	5	4	3	2	1	2	1	3	2	1	0	0	0

■ **Figure 1** Suffix array, its inverse, Φ , LCP array, and PLCP array of our running example string T .

n contiguous words from or to disk requires $\text{scan}(n) = \Theta(n/B)$ I/Os. Sorting n contiguous words requires $\text{sort}(n) = \Theta((n/B) \cdot \log_{M/B}(n/B))$ I/Os. For realistic values of n , B , and M , we stipulate that $\text{scan}(n) < \text{sort}(n) \ll n$.

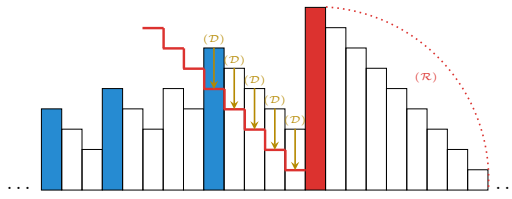
Text. Let Σ denote an integer alphabet of size $\sigma = |\Sigma| = n^{O(1)}$ for a natural number n . The alphabet Σ induces the *lexicographic order* \prec on the set of strings Σ^* . Let $|T|$ denote the length of a string $T \in \Sigma^*$. We write $T[j]$ for the j -th character of T , where $1 \leq j \leq n$. Given $T \in \Sigma^*$ consists of the concatenation $T = UVW$ for $U, V, W \in \Sigma^*$, we call U , V , and W a *prefix*, a *substring*, and a *suffix* of T , respectively. Given that the substring V starts at the i -th and ends at the j -th position of T , we also write $V = T[i..j]$ and $W = T[j+1..]$. In the following, we take an element $T \in \Sigma^*$ with $|T| = n$, and call it *text*. We stipulate that T ends with a sentinel $T[n] = \$ \notin \Sigma$ that is lexicographically smaller than every character of Σ .

Text Data Structures. Let SA denote the *suffix array* [24] of T . The entry $SA[i]$ is the starting position of the i -th lexicographically smallest suffix such that $T[SA[i]..] \prec T[SA[i+1]..]$ for all integers i with $1 \leq i \leq n-1$. Let ISA of T be the inverse of SA , i.e., $ISA[SA[i]] = i$ for every i with $1 \leq i \leq n$. The *Burrows-Wheeler transform (BWT)* [4] of T is the string BWT with $BWT[i] = T[n]$ if $SA[i] = 1$ and $BWT[i] = T[SA[i]-1]$ otherwise, for every i with $1 \leq i \leq n$. The *LCP array* is an array with the property that $LCP[i]$ is the length of the longest common prefix (LCP) of $T[SA[i]..]$ and $T[SA[i-1]..]$ for $i = 2, \dots, n$. For convenience, we stipulate that $LCP[1] := 0$. The array Φ is defined as $\Phi[i] := SA[ISA[i]-1]$, and $\Phi[i] := n$ in case that $ISA[i] = 1$. The *PLCP array* $PLCP$ stores the entries of LCP in text order, i.e., $PLCP[SA[i]] = LCP[i]$. Figure 1 illustrates the introduced data structures.

Idea for Using PLCP for Compression. Given a suffix $T[i..]$ starting at text position i , $PLCP[i]$ is the length of the longest common prefix of this suffix and the suffix $T[\Phi[i]..]$, which is its lexicographical predecessor among all suffixes of T . The longest common prefix of these two suffixes $T[i..]$ and $T[\Phi[i]..]$ is $T[i..i+PLCP[i]-1]$. The longest string among all these longest common prefixes (for each i with $1 \leq i \leq n$) is one of the longest re-occurring substrings in the text. Finding this longest re-occurring substring with $PLCP$ and Φ is the core idea of our compression algorithm. This algorithm produces a bidirectional scheme, which is defined as follows.

2 Compression Scheme

A *bidirectional scheme* [29] is defined by a factorization $F_1 \cdots F_b = T$ of a text T . A factor F_x is either a *referencing* factor or a *literal* factor. A referencing factor F_x is associated with a pair (src, ℓ) such that F_x and $T[src..src+\ell-1]$ are two different but possibly overlapping occurrences of the substring F_x in T . The pair (src, ℓ) and the text position src are called



■ **Figure 2** Visualization of Rules (\mathcal{D}) and (\mathcal{R}) being applied. Bars represent $PLCP$ values.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	a	b	a	b	b	a	b	a	b	a	b	b	a	b	b	a	a	b	a	b	a	\$
$PLCP$	4	5	4	3	4	5	5	7	6	5	4	3	2	1	2	1	3	2	1	0	0	0
$PLCP^1$	4	5	4	3	<u>3</u>	<u>2</u>	<u>1</u>	0	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>2</u>	1	3	2	1	0	0	0
$PLCP^2$	<u>1</u>	0	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	1	0	0	0	0	0	0	0	2	1	3	2	1	0	0	0
$PLCP^4$	1	0	0	0	0	0	1	0	0	0	0	0	0	0	2	1	0	<u>0</u>	<u>0</u>	0	0	0
$PLCP^3$	1	0	0	0	0	0	1	0	0	0	0	0	0	0	<u>0</u>	0	0	0	0	0	0	0

■ **Figure 3** Step-by-step execution of the $plcpcomp$ compression scheme (see Section 2) on $T = ababbabababbabbaababab\$$. We overwrite values of $PLCP$ according to Rules (\mathcal{D}) and (\mathcal{R}) . Each row $PLCP^i$ shows $PLCP$ after creating the i -th referencing factor starting at a position whose $PLCP$ entry is surrounded by a box. Changed entries according to Rules (\mathcal{D}) and (\mathcal{R}) are underlined.

reference and referred position, respectively. A factorization is *cycle-free*, i.e., references are not allowed to have cyclic dependencies. A factorization is called ξ -restricted for an integer $\xi \geq 2$ if each referencing factor F_x is at least ξ characters long (i.e., $\ell \geq \xi$).

A *unidirectional scheme* is a special case of a bidirectional scheme, with the restriction that the referred position of a referencing factor F_x must be smaller than the starting position of F_x . The most prominent example of a unidirectional scheme is the *LZ77* factorization, whose factorization is usually designed to be 2-restricted.

2.1 Coding

A bidirectional scheme codes the factors by substituting referencing factors with their associated references while keeping literal factors as strings. By doing so, the coding is a list whose x -th element is either a string (corresponding to a literal factor) or a reference representing the x -th factor ($1 \leq x \leq b$), which is referencing.

The $plcpcomp$ scheme and its predecessor, the $lcpcomp$ scheme [6], are bidirectional schemes. Both schemes are greedy, as they create a referencing factor equal to the longest re-occurring substring of the text that is not yet part of a factor. They differ in the selection of such a substring in case that there are multiple candidates with the same length. The $plcpcomp$ scheme can be computed with a rewritable $PLCP$ array and the following instructions:

1. Compute the set of *candidate positions* $\mathcal{C} := \{i \mid PLCP[i] \geq PLCP[j] \text{ for all text positions } j\}$.
2. Let dst be the leftmost position of all candidate positions \mathcal{C} . Terminate if $PLCP[dst] < \xi$.
3. Create a referencing factor by replacing $T[dst..dst + PLCP[dst] - 1]$ with the reference $(\Phi[dst], PLCP[dst])$.
4. Apply the following rules to ensure that we do not create overlapping factors (cf. Figure 2):
 - (\mathcal{D}) Decrease $PLCP[j] \leftarrow \min(PLCP[j], dst - j)$ for every $j \in [dst - PLCP[dst], dst]$.
 - (\mathcal{R}) Remove the factored positions by setting $PLCP[dst + k] \leftarrow 0$ for every $k \in [0, PLCP[dst])$.
5. Recurse with the modified $PLCP$.

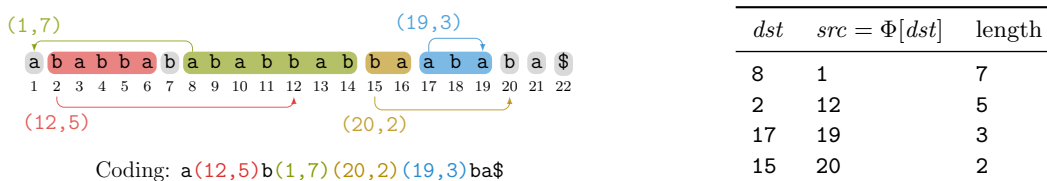


Figure 4 Coding of *plcpcomp* with $\xi = 2$. The factorization described in Figure 3 computes four referencing factors, listed in the table on the right. These factors are coded by their references. The factorization with *PLCP* in Figure 3 already determines the starting position and the lengths of all referencing factors (columns “*dst*” and “length” in the table). The referred positions are obtained using Φ (column “*src*”). The figure on the left illustrates factors as boxed substrings and the references as arrows from the starting positions of referencing factors to their respective referred positions.

An application of the above instructions on our running example is given in Figure 3. The coding is visualized in Figure 4. There and in the following figures, we fix $\xi := 2$.

2.2 Comparison to *lcpcomp*

The difference to *lcpcomp* [6] is that we fix *dst* to be the *leftmost* of all candidate positions in \mathcal{C} . [6] presented an algorithm computing the *lcpcomp* scheme in $\mathcal{O}(n \lg n)$ time with a heap storing the candidate positions ranked by their *PLCP* values. We can adapt this algorithm to compute the *plcpcomp* scheme by altering the order of the heap to rank the candidate positions first by their *PLCP* values (maximal *PLCP* values first) and second (in case of equal *PLCP* values) by their values themselves (minimal text positions first).

Since *lcpcomp* is cycle-free [6, Lemma 4] regardless of the selection of $dst \in \mathcal{C}$, we conclude that *plcpcomp* is also cycle-free, i.e., the substitution of substrings by references is reversible.

3 Computing the Factorization without Random Access

In this section, we present an algorithm for computing the *plcpcomp* scheme, which linearly scans *PLCP* without changing its contents. Instead of maintaining a heap storing all text positions ranked by their *PLCP* values, we compute the factorization by scanning the text sequentially from left to right. Although the algorithm will produce the *plcpcomp* factorization, it does not compute it in the order explained previously (starting with the longest factor). Instead, it first determines a subset of those substrings that define a referencing factor according to the *plcpcomp* scheme. The starting positions of these substrings have a *PLCP* value that is relatively large compared to their neighboring positions. We call those starting positions *peaks*.

Formally, we call a text position dst a *peak* if $PLCP[dst] \geq \xi$ and one of the following conditions holds:

- $dst = 1$,
- $PLCP[dst - 1] < PLCP[dst]$,¹ or
- there is a referencing factor ending at $dst - 1$.

A peak dst is called *interesting* if there is no text position j with $dst \in (j, j + PLCP[j])$ and $PLCP[j] \geq PLCP[dst]$. An interesting peak dst is called *maximal* if there is no interesting peak j with $j \in (dst, dst + PLCP[dst])$.

¹ A subset of the so-called *irreducible PLCP* entries [20, Lemma 4] have this property.

■ **Algorithm 1** Computation of *plpcomp* factors.

```

1  $L \leftarrow \emptyset$  // Step 1a
2 for  $dst = 1$  to  $n$  do // Step 1b
3   if  $dst$  is a maximal peak then // Step 2
4     create a referencing factor replacing  $T[dst \dots dst + PLCP[dst] - 1]$  // Step 3
5     apply Rule ( $\mathcal{D}$ ) to the peaks in  $L$ 
6     while  $L$  contains maximal peaks do
7        $j \leftarrow$  rightmost maximal peak in  $L$ 
8       create referencing factor replacing  $T[j \dots j + PLCP[j] - 1]$ 
9       apply Rules ( $\mathcal{D}$ ) and ( $\mathcal{R}$ ) to the peaks in  $L$ 
10      remove those elements of  $L$  that are no longer interesting peaks
11     $dst \leftarrow dst + PLCP[dst]$ 
12  if  $dst$  is an interesting peak then
13     $L \leftarrow L \cup \{dst\}$ 

```

Given an interesting peak dst , there is no text position j with $PLCP[j] \geq PLCP[dst]$ that becomes the starting position of a referencing factor containing $T[dst]$ (such that $PLCP[dst]$ cannot be removed according to Rule (\mathcal{R})). Given a maximal peak dst , there is additionally no text position j with $PLCP[j] > PLCP[dst]$ for which we apply Rule (\mathcal{D}) on $PLCP[dst]$ after factorizing $T[j \dots j + PLCP[j] - 1]$. Informally, we can determine whether a peak is interesting by looking at the $PLCP$ values *before* this peak, whereas we need to also look *ahead* for determining whether a peak is maximal. Given that there is at least one $PLCP$ entry with a value of at least ξ , we can find a maximal peak, since the leftmost position $\min \{i \in [1 \dots n] \mid PLCP[i] \geq PLCP[j] \text{ for all } j \text{ with } 1 \leq j \leq n\}$ among all positions with the highest $PLCP$ value is a maximal peak. The following lemma states that we can always factorize the leftmost maximal peak, regardless of whether the text has even higher peaks.

► **Lemma 1.** *If the text position dst is a maximal peak, then $T[dst \dots dst + PLCP[dst] - 1]$ is a referencing factor.*

Our preliminary algorithm consists of the following steps:

1. Scan $PLCP$ for the leftmost maximal peak dst .
2. Terminate if no such peak exists.
3. Create the referencing factor $T[dst \dots dst + PLCP[dst] - 1]$.
4. Apply Rules (\mathcal{R}) and (\mathcal{D}).
5. Interpret $T[1 \dots dst - 1]$ and $T[dst + PLCP[dst] \dots n]$ as two independent strings and recurse on each of them individually.

This algorithm produces the *plpcomp* scheme, because

- $T[dst \dots dst + PLCP[dst] - 1]$ is a referencing factor for each selected leftmost maximal peak dst according to Lemma 1, and
- the part $T[1 \dots dst - 1]$ can be factorized independently from how $T[dst + PLCP[dst] \dots n]$ is factorized, and vice versa. That is because, having already $T[dst \dots dst + PLCP[dst] - 1]$ factorized, we can no longer create a factor that covers a text position in the range $[dst \dots dst + PLCP[dst] - 1]$.

Hence, we can factorize $T[1 \dots dst - 1]$ without considering the factorization of the rest of the text to produce the correct *plpcomp* scheme. Figure 5 illustrates the computation of the *plpcomp* factorization with this algorithm.

However, as the algorithm overwrites entries of $PLCP$, it is not yet satisfying. A rewritable $PLCP$ array would have to be kept in RAM, costing us $n \lg n$ bits of space if we require constant time read and write access. Instead of keeping $PLCP[1 \dots dst - 1]$ in RAM, we now

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	a	b	a	b	b	a	b	a	b	a	b	b	a	b	b	a	a	b	a	b	a	\$
$PLCP$	4	5	4	3	4	5	5	7	6	5	4	3	2	1	2	1	3	2	1	0	0	0
$PLCP^1$	<u>1</u>	0	0	0	0	0	0	5	7	6	5	4	3	2	1	2	1	3	2	1	0	0
$PLCP^1$	1	0	0	0	0	0	<u>1</u>	0	0	0	0	0	0	0	0	2	1	3	2	1	0	0
$PLCP^2$	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	3	2	1	0	0
$PLCP^3$	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5 Step-by-step execution of our *plcpcomp* algorithm on $T = \text{ababbabababbabbaababa\$}$. While the instructions of the scheme (cf. Section 2) always replace the factor starting at a position with the maximal *PLCP* value (cf. Figure 3), our algorithm described in Section 3 creates a factor at the leftmost maximal peak. Our algorithm computes the same factorization as described in the *plcpcomp* scheme, but in different order.

show that it suffices to manage only the *PLCP* values of the *interesting peaks*. For that, we enhance the search of the leftmost maximal peak by replacing the first step of the algorithm by the following instructions:

- 1a. Create an empty list of peaks L .
- 1b. Scan T from left to right until a maximal peak dst is found. While doing so, insert all visited interesting peaks into L .

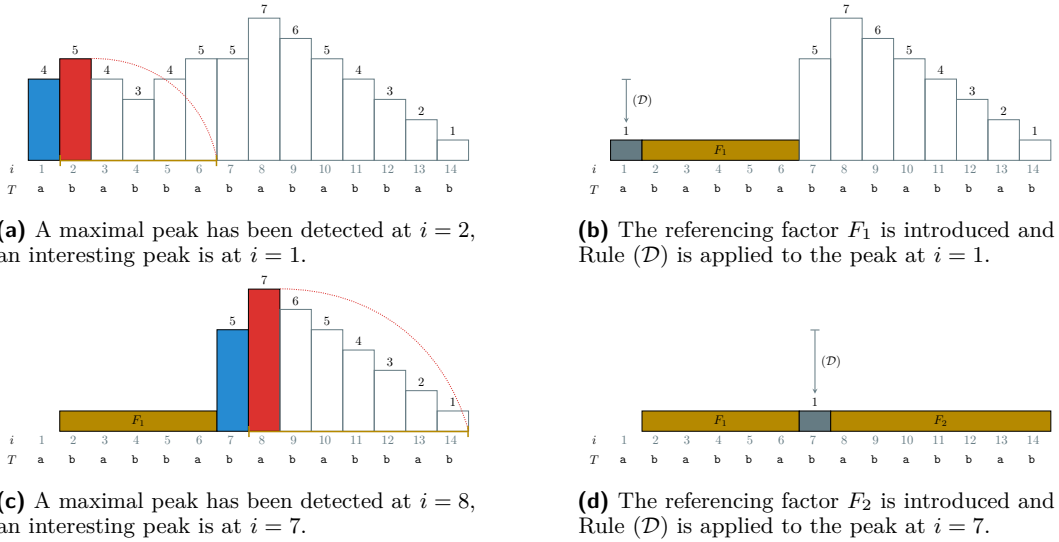
Another alternation is that we apply Step 4 only to the peaks stored in L . There, we scan L from right to left while applying Rule (\mathcal{D}) and removing all elements that are no longer interesting peaks. The modified algorithm is sketched as pseudo code in Algo. 1.

► **Example 2.** Figure 6 illustrates Algo. 1 on the prefix $T[1..14] = \text{ababbabababbab}$ of our running example in three steps. The peaks at positions 1 and 2 are interesting. Since the peak at position 2 is the highest interesting peak, it is the maximal peak, which is detected after scanning $PLCP[1..6]$ (Figure 6a). In the second step (Figure 6b), the referencing factor F_1 is introduced, which starts at this maximal peak. As a consequence, Rule (\mathcal{D}) is applied to the only peak stored in L , the one at position 1. However, because the *PLCP* value 1 is below the threshold $\xi = 2$, the peak at position 1 is removed from L . Since L is then empty, we proceed with the next scan for a maximal peak starting from position 7. By definition, the peak at position 7 becomes interesting. The next maximal peak is detected at position 8 (Figure 6c). The factor F_2 (Figure 6d) is introduced, and Rule (\mathcal{D}) is applied to the peak at position 7. Its *PLCP* value drops below our threshold and thus it is removed from L . Finally, the prefix $T[1..14]$ has been processed.

In Algo. 1, we omit all other peaks that are not stored in L when applying Rules (\mathcal{D}) and (\mathcal{R}). Thus, it suffices to maintain the *PLCP* value of each peak in L in an extra list instead of maintaining a complete rewritable *PLCP* array. In the following, we prove why this omission still produces the correct factorization (Lemma 5). For that, we show that we can produce the *plcpcomp* factors contained in $T[1..dst + PLCP[dst] - 1]$ only with the *PLCP* values of the peaks stored in L (first recursive call). We start with the following property of L :

► **Lemma 3.** *The positions stored in L are in strictly ascending order with respect to their LCP values.*

Next, we examine the result of creating the referencing factor $T[dst..dst + PLCP[dst] - 1]$ starting at the maximal peak dst . After creating this factor, the *PLCP* values of peaks near dst can be decreased. However, this causes at most one new peak as can be seen by the following lemma.



■ **Figure 6** Execution of our algorithm of Section 3 computing the *plcpcomp* compression scheme on $T = \text{ababbababbabbaababa}\$$. Due to limited space, we only illustrate the processing of the prefix $T[1..14]$ in three steps (explained in Example 2). The vertical bars represent the *PLCP* array, with the corresponding values written above, in text order from left ($i = 1$) to right ($i = 14$). The shaded vertical bars represent the (current) *PLCP* value of an interesting peak. Horizontal bars represent (referencing) factors. In (b), the factor F_1 , starting at position 2, is displayed as the maximal peak being *tipped over* to the right.

► **Lemma 4.** *Applying Rules (\mathcal{D}) and (\mathcal{R}) after creating a referencing factor F_x does not cause new peaks, with the only possible exception of the position succeeding the end of F_x .*

Since Rule (\mathcal{D}) decreases at most the values of $PLCP[dst - PLCP[dst]..dst - 1]$, the highest peak dst' in $PLCP[1..dst - 1]$ is an interesting peak that is either

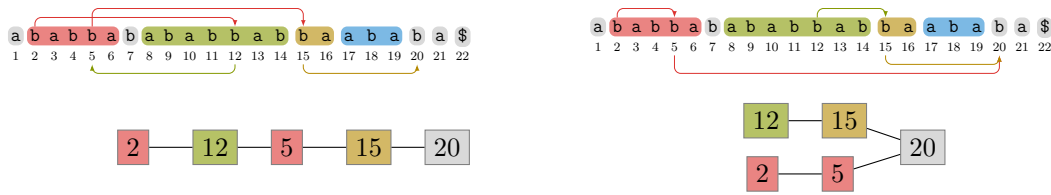
- in the interval $[dst - PLCP[dst]..dst - 1]$, or,
- in the case that all interesting peaks in $[dst - PLCP[dst]..dst - 1]$ are no longer interesting after decreasing their *PLCP* values, the rightmost peak preceding $dst - PLCP[dst]$ (whose *PLCP* value is equal to the *PLCP* value of the last peak removed from L in Step 4).

We can locate dst' while applying Rule (\mathcal{D}) as a result of creating the factor starting at dst . After locating dst' , we apply the following steps recursively:

1. Substitute $T[dst'..PLCP[dst'] - 1]$ with a reference, because it is the highest peak in $T[1..dst - 1]$.
2. If $dst'' := dst' + PLCP[dst']$ with $PLCP[dst''] \geq \xi$ was not a peak, then dst'' becomes an interesting peak. In this case, substitute dst' with dst'' in L to preserve the order in L . Otherwise, remove dst' from L .
3. Split L into two sub-lists:
 - one containing text positions of the range $[1..dst' - 1]$, and
 - the other containing text positions of the range $[dst' + PLCP[dst']..dst - 1]$.
4. Recurse on each of the two sub-lists, i.e., find the highest peak in each sub-list and substitute it.

This recursion is more efficient than the while-loop described in Lines 6 to 10 of Algo. 1.

► **Lemma 5.** *The algorithm emits a valid *plcpcomp* factorization of $T[1..dst + PLCP[dst] - 1]$.*



■ **Figure 7** Pointer jumping applied to references. Suppose that our example text is represented by the coding described in Figure 4. To extract the character $T[2]$, we need to resolve the reference $(12, 5)$, which has a depth of three (*bottom left* figure). In case that we split all references into references of length one, we can reduce the depth of the reference associated with $T[2]$ by pointer jumping (*right* figure). The order in which this technique is applied to the references has an impact on the resulting references. Here, we assumed that we can apply this technique *in parallel*.

After factorizing $T[1 \dots dst + PLCP[dst] - 1]$, we proceed with Algo. 1 on the remaining text $T[dst + PLCP[dst] \dots]$ to compute the factorization of the entire text. It is left to explain how this algorithm can be adapted to the EM model efficiently.

3.1 Factorization in External Memory

Having the text, $PLCP$, and Φ stored as files in EM, we can compute the *plcpcmp* scheme in three sequential scans over n tuples and one sort operation:

1. Proceed with Algo. 1 to find pairs $(dst, \ell = PLCP[dst])$ representing referencing factors $T[dst \dots dst + \ell]$ by scanning $PLCP$.
2. Sort these pairs in ascending order of their dst components (i.e., in text order).
3. Simultaneously scan this sorted list of pairs and Φ to compute triplets of the form $(dst, src = \Phi[dst], \ell)$, where the second component is the referred position of the referencing factor $T[dst \dots dst + \ell - 1]$.
4. Finally, scan simultaneously the list of references and T to replace each substring $T[dst \dots dst + \ell - 1]$ by the reference (src, ℓ) on reading the triplet (dst, src, ℓ) .

The pairs emitted during the $PLCP$ scan (Step 1) can be stored and then sorted in EM. The references computed by the second scan can be written to disk for the final scan, which computes the *plcpcmp* scheme of T sequentially. By doing so, no random access is required on the list of references.

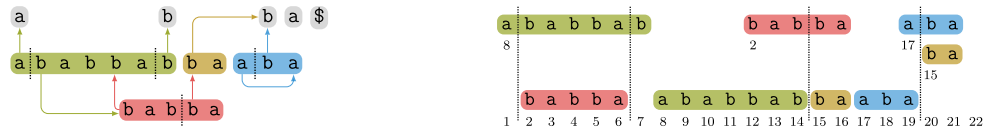
During the $PLCP$ scan, the list L can also be maintained on disk efficiently: until a maximal peak is found, we only append peaks to L . For our experiments, we store L in RAM, as the number of elements was much lower than the upper bound $\mathcal{O}(\min(\sqrt{n \lg n}, r))$ where r is the number of BWT runs (see the full version of this paper).

Once a maximal peak dst has been found and a reference (dst, ℓ) is emitted, we scan over L sequentially (a) to apply Rules (\mathcal{D}) and (\mathcal{R}) and (b) to find a remaining maximal peak, if any, in the process. We then repeat this process until there are no more maximal peaks in L . In practice, we scan the last elements of L linearly from right to left, since only the last interesting peaks need to be updated.

4 Decompression

The task of decompressing a bidirectional scheme is to *resolve* each reference (src_i, ℓ_i) of a referencing factor $T[dst_i \dots dst_i + \ell_i - 1]$, i.e., to copy the characters from $T[src_i \dots src_i + \ell_i - 1]$ to $T[dst_i \dots dst_i + \ell_i - 1]$.

41:10 Bidirectional Text Compression in External Memory



■ **Figure 8** The dependency graph (*left*) and its EM representation (*right*) of the factorization given in Figure 4. The multi-dependent factors of length seven and five have a cyclic dependency. The EM representation of the graph described in Section 4 consists of two copies of the list of all referencing factors, sorted by their source position (*top*) as well as sorted by their destination (*bottom*).

A unidirectional scheme can be decompressed by scanning linearly over the compressed input from left to right. In that scenario, references can be resolved easily because they always refer to already decompressed parts of the text [2]. This property does not hold for a bidirectional scheme in general, as a reference can refer to a part of the text that again corresponds to a reference.

► **Definition 6 (Dependency Graph).** *Given a bidirectional factorization $F_1 \cdots F_b = T$, we model its references as a directed graph G with $V = \{v_1, \dots, v_b\}$ such that there is a 1-to-1 correlation between nodes v_i and factors F_i . We add a directed edge (v_i, v_j) from v_i to v_j with $i \neq j$ iff F_i refers to at least one character in the factor F_j . We put these edges into a set E to form a graph $G := (V, E)$ that has only literal factors as sinks. A node v_i can have more than one out-going edge if the referred substring is covered by multiple factors; in this case, we say v_i is multi-dependent and call the set of its out-going edges a multi-dependency. The dependency graph of our example from Figure 4 can be seen in Figure 8.*

Bidirectional decompressors face two challenges arising from this graph structure:

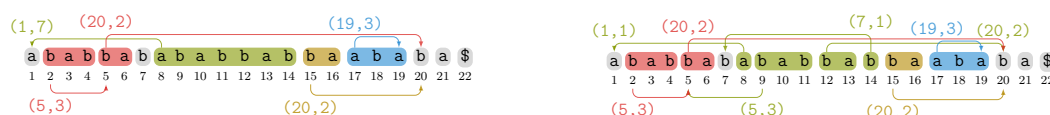
- The existence of multi-dependent nodes disallows efficient tree-based approaches.
- Long dependency chains may affect the time and space complexity of decompression.

Our compression scheme splits multi-dependencies into single dependencies and deploys the pointer jumping technique [14, Sect. 2.2] for dependency resolution. After the resolution we obtain a dependency graph in which each reference is single-dependent on a literal factor. Then the text can be trivially recovered with $\text{sort}(n)$ I/Os. The details are as follows.

Let G be the dependency graph of the factorization $T = F_1 \cdots F_b$. For now we assume that all factors are single-dependent, i.e., each node v representing a referencing factor has exactly one outgoing edge $(v, p(v))$. For all other nodes (representing literal factors) we define $p(v) := v$. Clearly, G forms a forest in which each tree is rooted in a literal factor. When applying the pointer jumping technique, we take each referencing factor and attach it to the parent of its parent (cf. Figure 7). Given that G' is the resulting graph with $p'(v) = p(p(v))$, we thereby halve the depth, i.e., $d(G') = \lceil d(G)/2 \rceil$ if $d(G) \geq 2$, where $d(G)$ denotes the maximum depth of a tree in G . Hence, after $\Theta(\lg d(G))$ iterations all indirect references are resolved and have been replaced by direct references to literal factors.

If we allow multi-dependencies, pointer jumping is only possible for single-dependent nodes. To apply pointer jumping, we split each multi-dependent reference into the smallest possible set of single-dependent references. A split is introduced ad-hoc each time it is required for a pointer jump. The details of the splitting are discussed in the full version of this paper.

We first construct a representation of the dependency graph consisting of two EM vectors called **requests** and **factors**. Intuitively, each request (child) sends a request message to the first factor it refers to (parent). Addressing is implemented indirectly in terms of



■ **Figure 9** Split-Strategy of *EM-PJ* applied to the first (*left* figure) and second (*right* figure) referencing factor of the factorization given in Figure 4. *EM-PJ* splits up references in a minimal number of sub-references on which the pointer jumping technique can be applied. The *left* figure shows such an application to the reference of the leftmost referencing factor that is split into two sub-references. The first and second sub-reference receive new referred positions based on the referred positions of the second and third referencing factors, respectively. In the *right* figure, we split up the next reference (1,7) in four sub-references, where the first and last sub-reference refer to literal factors.

text positions rather than factor indices. For each reference (src, ℓ) corresponding to a factor $F_i = T[dst..dst + \ell - 1]$, we push $\langle dst, \ell, src \rangle$ into `requests` and $\langle src, \ell, dst \rangle$ into `factors`. We omit literal factors, since the lack of a reference in `factors` for a certain text position indicates the presence of a literal factor.

Subsequently, we sort² both vectors independently, bringing the messages in `requests` and the recipients in `factors` into the same order. On the right side of Figure 8 we see a visualization of the lists (after the initial sorting) for our running example. We augment `requests` with an initially empty EM priority queue `PQSPLIT`. In the following, after processing a factor F_i , we write F_i either to a vector `result` if it refers to literal factors, or to a vector `nextRequests` otherwise: Let $\langle dst, \ell, src \rangle$ be the smallest unprocessed request of a factor F_i received via `requests` or `PQSPLIT`. If it originates from `requests`, we advance `requests`'s read pointer for the next iteration, otherwise we dequeue the top element from `PQSPLIT`. We process the read request $\langle dst, \ell, src \rangle$ depending on the following cases (cf. Figure 9):

- **Jump** The request is completely covered by parent F_j in `factors`. In this case, we substitute F_i 's reference according to F_j and push it into `nextRequests` to be processed in the next iteration.
- **Finalize** No parent (partially) overlapping with F_i is available in `factors`. Then we know that F_i points to a substring contained in literal factors. We finalize F_i by pushing it into `result`.
- **Split** A prefix of F_i is contained in the parent F_j or points to literals. Let $\ell' < \ell$ be the length of the longest such prefix. Then split F_i into a prefix F_i^P of length ℓ' and a suffix F_i^S of length $\ell - \ell'$. By construction, either case “Jump” or case “Finalize” is applicable to F_i^P , and we execute it directly. Then we push $\langle src + \ell', \ell - \ell', dst + \ell' \rangle$ representing F_i^S into `PQSPLIT` to process it later within the same iteration. Observe that F_i can be split multiple times during the same iteration.

If `nextRequests` is not empty, we sort it and recurse by processing `nextRequests` and the (unaltered) `factors` simultaneously as before. With these steps, we obtain the final result:

► **Theorem 7.** *Let $F_1 \cdots F_b = T$ be a ξ -restricted bidirectional scheme, and $d(G) < b$ be the depth of T 's dependency graph G . Then *EM-PJ* requires $\mathcal{O}(\lg(d(G)) \text{sort}(n/\xi))$ I/Os.*

² To sort tuples we always use lexicographic order, i.e., we order tuples by their first unequal elements.

■ **Table 1** Empirical entropies of our data sets. The alphabet sizes of all instances are 242 and 4 for `commoncrawl` and `dna`, respectively.

commoncrawl								
prefix length	H_0	H_1	H_2	H_3	H_4	H_5	H_6	H_7
16 GiB	5.99165	4.26109	3.48920	2.94113	2.42738	2.01886	1.64558	1.35130
32 GiB	5.99145	4.26160	3.49006	2.94411	2.43471	2.03284	1.66737	1.37798
64 GiB	5.99119	4.26209	3.49100	2.94669	2.44088	2.04409	1.68482	1.40001
128 GiB	5.99177	4.26148	3.49055	2.94684	2.44231	2.04753	1.69087	1.40839
dna								
prefix length	H_0	H_1	H_2	H_3	H_4	H_5	H_6	H_7
16 GiB	1.9715	1.94676	1.93166	1.92232	1.91167	1.89491	1.87101	1.84585
32 GiB	1.97128	1.94561	1.93201	1.92421	1.91507	1.90190	1.88270	1.86160
64 GiB	1.97067	1.94506	1.93145	1.92424	1.91588	1.90445	1.88763	1.86889
128 GiB	1.97528	1.95010	1.93873	1.93273	1.92486	1.91341	1.89601	1.87634

5 Practical Evaluation

Experimental Setup. Our experiments are conducted on a machine with 16 GiB of RAM³, eight Hitachi HUA72302 hard drives with 1.8 TiB, two Samsung SSD 850 SSDs with 465.8 GiB, and an Intel Xeon CPU i7-6800K. The operating system is a 64-bit version of Ubuntu Linux 16.04. We implemented *plcpcomp*⁴ in the version 1.4.99 (development snapshot) of the STXXL library [5]. We compiled the source code with the GNU g++ 7.4 compiler.

Text Collections. We conduct our experiments on two texts of different alphabet sizes and repetitiveness (cf. Table 1):

- COMMONCRAWL: A crawl of web pages with an alphabet size of 242 collected by the commoncrawl organization.
- DNA: DNA sequences with an alphabet size of 4 extracted from FASTA files.

Algorithms. We compare *plcpcomp* against *EM-LPF* [17] by Kärkkäinen et al., which is an EM algorithm computing the *LZ77* factorization by constructing the *LPF* array. In addition to the input text, it requires *SA* and *LCP*.

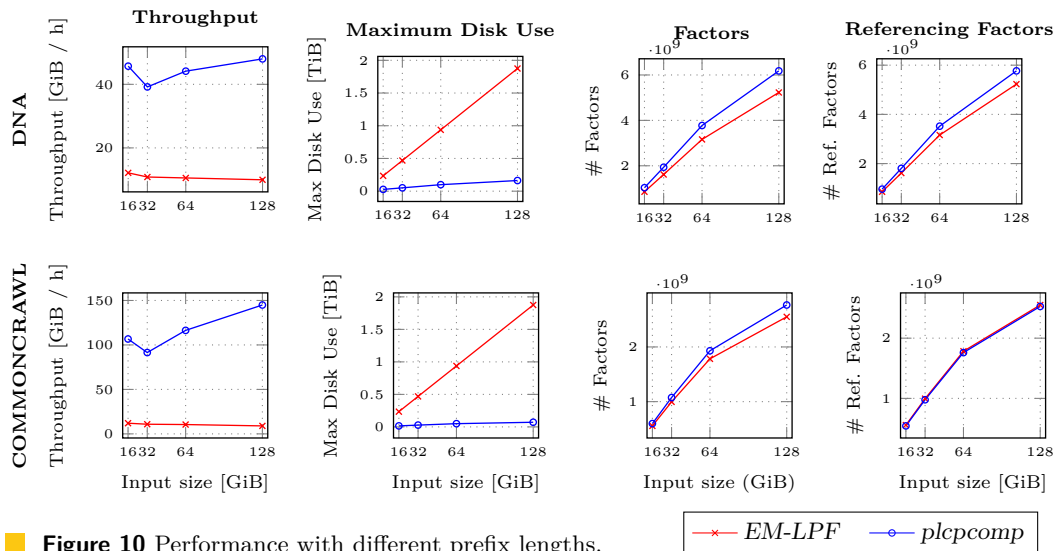
In early experiments with *EM-LZscan* [17], it became clear that its throughput on the text collection we use is nowhere near competitiveness with *EM-LPF* and *plcpcomp*. Therefore, it is not considered in our experiments. Semi-external *LZ77* algorithms like *SE-KKP* [17] storing the text or parts of the text in RAM have not been considered.

Data Structures. Currently, the fastest way to compute the data structures *PLCP* and Φ in EM is to compute *BWT* from *SA* with the parallel EM algorithm *pEM-BWT* by Kärkkäinen and Kempa⁵, and use it for computing *PLCP* with the parallel EM construction algorithm of [16]. We modified the source code of the latter to also produce Φ as a side product.

³ In order to avoid swapping, each experiment was conducted with a limit of 14 GiB of RAM.

⁴ Available at <https://github.com/tudocomp/tudocomp>.

⁵ https://www.cs.helsinki.fi/u/dkempa/pem_bwt.html



■ **Figure 10** Performance with different prefix lengths.

For *EM-LPF*, we additionally need to convert *PLCP* to *LCP* by a scan over *SA* and a subsequent sort step. This is currently the fastest approach for obtaining *LCP*, as other approaches building *LCP* directly from *SA* like [15] are slower.

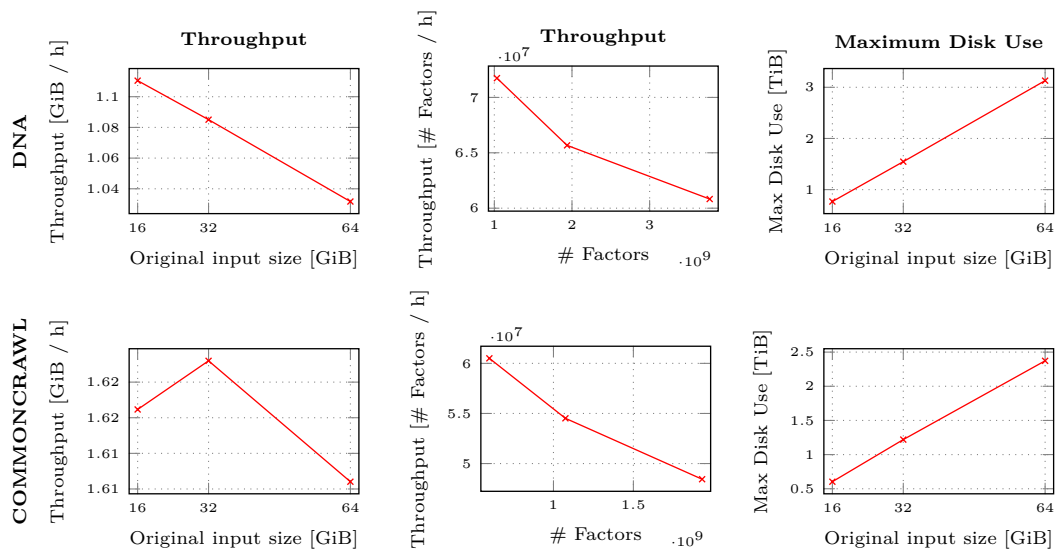
Consequently, both contestants need (directly or indirectly) *SA*. However, it takes a considerable amount of time to construct it with EM algorithms on a single machine (e.g., with pSAScan [18]). To put the focus on the comparison between *EM-LPF* and *plcpcomp*, we do not take into account the construction of *SA* and *LCP* when measuring running times.

Measurements and Results. Our experiments measure the throughput, the maximum hard disk usage, and the number of referencing factors, for *EM-LPF* and *plcpcomp* for 2^k GiB prefixes ($4 \leq k \leq 7$) of our data sets DNA and COMMONCRAWL. We collected the median of three iterations and present the results in Figure 10. The plots show that *plcpcomp* is magnitudes faster on both data sets (cf. plots “Throughput”). The reason for this could be that the disk accesses of *EM-LPF* scale much worse than those of *plcpcomp* (cf. plots “Maximum Disk Use”). We point out that *plcpcomp* is already faster than the step for computing *LCP* from *PLCP* and *SA*. Regarding the number of factors, *plcpcomp* is on par with *LZ77* (rightmost plots), producing, relatively speaking, slightly more factors. Our decompression requires multiple sorts of factor sets depending on the maximum depth of (a tree in) the dependency graph induced by the factorization. Therefore, it is not surprising that it is a lot slower than the comparatively simple compression.

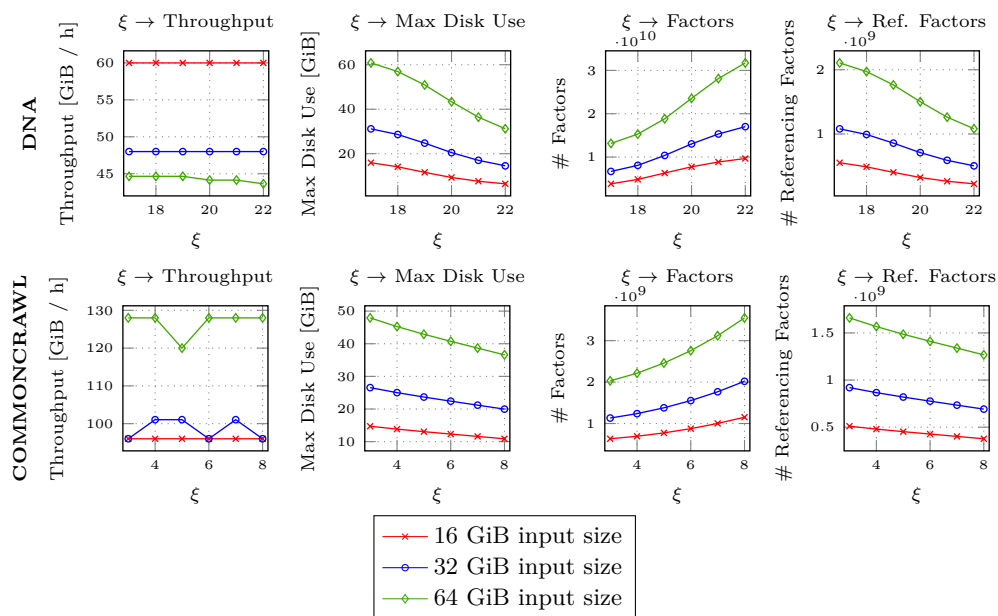
Furthermore, and for the same reason, our decompression expectedly runs orders of magnitudes slower than the external memory Lempel-Ziv decoder of [2], which is why we do not do a more detailed performance comparison here.

Decompression. We ran our decompressor implementation on the *plcpcomp* codings of our datasets. Plots of the scaling experiments are shown in Figure 11. As the decompression algorithm is superlinear, the throughput is decreasing with increasing text size. However, comparing the results for the 32GiB and 64GiB commoncrawl decompression, the throughput only decreases by 1%. The throughput between the 32GiB and 64 GiB DNA decompression differs by only 5%. The maximum external memory allocation rises linearly with increasing text size.

41:14 Bidirectional Text Compression in External Memory



■ **Figure 11** Performance of the decompression with different prefix lengths.



■ **Figure 12** Evaluation of *plpcomp* with different threshold values ξ .

In Figure 12, we measured the impact of the choice of ξ on the compressed output and the decompression algorithm of our datasets. For larger values of ξ , *plpcomp* creates less referencing factors, but the total number of factors increases (as we obtain much more literal factors). Having less referencing factors, the decompression needs less disk space.

Our decompression requires multiple sorting steps on the factor lists such as *requests* (cf. Section 4). The number of these steps depend on the maximum depth of (a tree in) the dependency graph induced by the factorization. Therefore, it is not surprising that the decompressor is magnitudes slower than the comparatively simple compression algorithm.

Furthermore, and for the same reason, our decompression (expectedly) runs slower than the external memory Lempel-Ziv decoder of [2], which is why we skip a more detailed performance comparison here.

6 Conclusions

We presented *plpcomp*, the first external memory bidirectional compression algorithm, and showed its practicality by performing experiments on very large data sets, using only very limited RAM. We also presented a decompression algorithm in external memory, which can decode the output of any bidirectional compression scheme (not only *plpcomp*). Possible future steps include relating the number of factors of *plpcomp* to the minimal number of factors in a bi- or unidirectional compression scheme, evaluating the whole compression chain by also experimenting on *codings* of the output of *plpcomp* (similar to [6]), and improving the performance of the decompression algorithm.

References

- 1 Alok Aggarwal and Jeffrey Scott Vitter. The Input/Output Complexity of Sorting and Related Problems. *Commun. ACM*, 31(9):1116–1127, 1988.
- 2 Djamal Belazzougui, Juha Kärkkäinen, Dominik Kempa, and Simon J. Puglisi. Lempel-Ziv decoding in external memory. In *Proc. SEA*, volume 9685 of *LNCS*, pages 63–74, 2016.
- 3 Timothy C. Bell. Better OPM/L Text Compression. *IEEE Trans. Communications*, 34(12):1176–1182, 1986.
- 4 M. Burrows and D. J. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994.
- 5 Roman Dementiev, Lutz Kettner, and Peter Sanders. STXXL: standard template library for XXL data sets. *Softw., Pract. Exper.*, 38(6):589–637, 2008.
- 6 Patrick Dinklage, Johannes Fischer, Dominik Köppl, Marvin Löbel, and Kunihiko Sadakane. Compression with the tudocomp Framework. In *Proc. SEA*, volume 75 of *LIPICs*, pages 13:1–13:22, 2017. [arXiv:1702.07577](https://arxiv.org/abs/1702.07577).
- 7 Paolo Ferragina, Travis Gagie, and Giovanni Manzini. Lightweight Data Indexing and Compression in External Memory. *Algorithmica*, 63(3):707–730, 2012.
- 8 Paolo Ferragina, Igor Nitto, and Rossano Venturini. On the Bit-Complexity of Lempel-Ziv Compression. *SIAM J. Comput.*, 42(4):1521–1541, 2013.
- 9 Johannes Fischer, Tomohiro I, Dominik Köppl, and Kunihiko Sadakane. Lempel-Ziv Factorization Powered by Space Efficient Suffix Trees. *Algorithmica*, 80(7):2048–2081, 2018.
- 10 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. On the Approximation Ratio of Lempel-Ziv Parsing. In *Proc. LATIN*, volume 10807 of *LNCS*, pages 490–503, 2018.
- 11 J. K. Gallant. *String compression algorithms*. PhD thesis, Princeton University, 1982.
- 12 Keisuke Goto and Hideo Bannai. Simpler and Faster Lempel Ziv Factorization. In *Proc. DCC*, pages 133–142, 2013.
- 13 Keisuke Goto and Hideo Bannai. Space Efficient Linear Time Lempel-Ziv Factorization for Small Alphabets. In *Proc. DCC*, pages 163–172, 2014.

- 14 Joseph JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.
- 15 Juha Kärkkäinen and Dominik Kempa. LCP array construction in external memory. *ACM Journal of Experimental Algorithmics*, 21(1):1.7:1–1.7:22, 2016.
- 16 Juha Kärkkäinen and Dominik Kempa. Engineering External Memory LCP Array Construction: Parallel, In-Place and Large Alphabet. In *Proc. SEA*, volume 75 of *LIPICs*, pages 17:1–17:14, 2017.
- 17 Juha Kärkkäinen, Dominik Kempa, and Simon J. Puglisi. Lempel-Ziv parsing in external memory. In *Proc. DCC*, pages 153–162, 2014.
- 18 Juha Kärkkäinen, Dominik Kempa, and Simon J. Puglisi. Parallel External Memory Suffix Sorting. In *Proc. CPM*, volume 9133 of *LNCS*, pages 329–342, 2015.
- 19 Juha Kärkkäinen, Dominik Kempa, and Simon John Puglisi. Lightweight Lempel-Ziv Parsing. In *Proc. SEA*, volume 7933 of *LNCS*, pages 139–150. Springer, 2013.
- 20 Juha Kärkkäinen, Giovanni Manzini, and Simon J. Puglisi. Permuted Longest-Common-Prefix Array. In *Proc. CPM*, volume 5577 of *LNCS*, pages 181–192, 2009.
- 21 Dominik Kempa and Dmitry Kosolobov. LZ-end parsing in compressed space. In *Proc. DCC*, pages 350–359, 2017.
- 22 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In *Proc. STOC*, pages 827–840, 2018.
- 23 Sebastian Kreft and Gonzalo Navarro. LZ77-like compression with fast random access. In *Proc. DCC*, pages 239–248, 2010.
- 24 Udi Manber and Eugene W. Myers. Suffix Arrays: A New Method for On-Line String Searches. *SIAM J. Comput.*, 22(5):935–948, 1993.
- 25 Markus Mauer, Timo Beller, and Enno Ohlebusch. A Lempel-Ziv-style Compression Method for Repetitive Texts. In *Proc. PSC*, pages 96–107, 2017.
- 26 Ryosuke Nakamura, Shunsuke Inenaga, Hideo Bannai, Takashi Funamoto, Masayuki Takeda, and Ayumi Shinohara. Linear-Time Text Compression by Longest-First Substitution. *Algorithms*, 2(4):1429–1448, 2009.
- 27 Akihiro Nishi, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. $\mathcal{O}(n \log n)$ -time text compression by LZ-style longest first substitution. In *Proc. PSC*, pages 12–26, 2018.
- 28 Takaaki Nishimoto and Yasuo Tabei. LZRR: LZ77 parsing with right reference. *arXiv 1812.04261*, 2018. [arXiv:1812.04261](https://arxiv.org/abs/1812.04261).
- 29 James A. Storer and Thomas G. Szymanski. Data compression via textural substitution. *J. ACM*, 29(4):928–951, 1982.
- 30 Vladimir Yanovsky. ReCoil - an algorithm for compression of extremely large datasets of DNA data. *Algorithms for Molecular Biology*, 6(23):1–9, 2011.

Bisection of Bounded Treewidth Graphs by Convolutions

Eduard Eiben

Department of Informatics, University of Bergen, Norway
eduard.eiben@uib.no

Daniel Lokshtanov

Department of Computer Science, US Santa Barbara, United States
daniello@ucsb.edu

Amer E. Mouawad

Department of Computer Science, American University of Beirut, Lebanon
aa368@aub.edu.lb

Abstract

In the BISECTION problem, we are given as input an edge-weighted graph G . The task is to find a partition of $V(G)$ into two parts A and B such that $||A| - |B|| \leq 1$ and the sum of the weights of the edges with one endpoint in A and the other in B is minimized. We show that the complexity of the BISECTION problem on trees, and more generally on graphs of bounded treewidth, is intimately linked to the $(\min, +)$ -CONVOLUTION problem. Here the input consists of two sequences $(a[i])_{i=0}^{n-1}$ and $(b[i])_{i=0}^{n-1}$, the task is to compute the sequence $(c[k])_{k=0}^{n-1}$, where $c[k] = \min_{i=0, \dots, k} (a[i] + b[k - i])$.

In particular, we prove that if $(\min, +)$ -CONVOLUTION can be solved in $O(\tau(n))$ time, then BISECTION of graphs of treewidth t can be solved in time $O(8^t t^{O(1)} \log n \cdot \tau(n))$, assuming a tree decomposition of width t is provided as input. Plugging in the naive $O(n^2)$ time algorithm for $(\min, +)$ -CONVOLUTION yields a $O(8^t t^{O(1)} n^2 \log n)$ time algorithm for BISECTION. This improves over the (dependence on n of the) $O(2^t n^3)$ time algorithm of Jansen et al. [SICOMP 2005] at the cost of a worse dependence on t . “Conversely”, we show that if BISECTION can be solved in time $O(\beta(n))$ on edge weighted trees, then $(\min, +)$ -CONVOLUTION can be solved in $O(\beta(n))$ time as well. Thus, obtaining a sub-quadratic algorithm for BISECTION on trees is extremely challenging, and could even be impossible. On the other hand, for *unweighted* graphs of treewidth t , by making use of a recent algorithm for BOUNDED DIFFERENCE $(\min, +)$ -CONVOLUTION of Chan and Lewenstein [STOC 2015], we obtain a sub-quadratic algorithm for BISECTION with running time $O(8^t t^{O(1)} n^{1.864} \log n)$.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms; Theory of computation \rightarrow Graph algorithms analysis

Keywords and phrases bisection, convolution, treewidth, fine-grained analysis, hardness in P

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.42

1 Introduction

A *bisection* of a graph G is a partition of $V(G)$ into two parts A and B such that $||A| - |B|| \leq 1$. The *weight* of a bisection (A, B) of an edge-weighted graph G is the sum of the weights of all edges with one endpoint in A and the other in B . In the BISECTION problem the task is to find a minimum weight bisection in an edge-weighted graph G given as input. The problem can be seen as a version of MINIMUM CUT with a balance constraint on the sizes of two sides of the cut. While MINIMUM CUT is solvable in polynomial time, BISECTION is one of the classic examples of NP-complete problems [15]. BISECTION has been studied extensively from the perspective of approximation algorithms [14, 13, 18, 21], parameterized algorithms [7, 11, 22] heuristics [6, 8] and average case complexity [5].



© Eduard Eiben, Daniel Lokshtanov, and Amer E. Mouawad;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 42; pp. 42:1–42:11

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper we consider BISECTION when the input graph is required to be a tree, or more generally a graph with treewidth at most t . For trees, an $O(n^3)$ time algorithm was given by MacGregor [20] already in 1978, this was improved to a parallel algorithm running in time $O(\log^2 n \log \log n)$ on $O(n^2)$ processors by Goldberg and Miller [16]. This corresponds to a sequential algorithm running in time $O(n^2 \log^2 n \log \log n)$. For graphs of bounded treewidth Jansen et al. [17] gave an algorithm that solves BISECTION in time $O(2^t n^3)$ if a tree decomposition of width t is given as input.

The majority of natural graph problems are solvable in linear time on trees and bounded treewidth graphs (see e.g. Courcelle’s theorem [10]). Thus, it is quite natural to ask whether the dependence on n in the algorithm of Jansen et al. [17] could be improved to linear. Our first result goes “half the way” from Jansen et al.’s cubic algorithm to a linear time one, and matches (in fact slightly improves) the fastest known algorithm for BISECTION on trees¹.

► **Theorem 1.** *There is an algorithm that, given an edge-weighted graph G on n vertices together with a tree decomposition of G of width at most t , computes a minimum weight bisection of G in time $\mathcal{O}(8^t \cdot t^5 \cdot n^2 \cdot \log n)$.*

Our algorithm crucially uses the $(\min, +)$ -convolution operation. The $(\min, +)$ -convolution of two number sequences $(a[i])_{i=0}^{n-1}$ and $(b[i])_{i=0}^{n-1}$ is a sequence $(c[i])_{i=0}^{n-1}$, where $c[k] = \min_{i=0, \dots, k} (a[i] + b[k - i])$. In the $(\min, +)$ -CONVOLUTION problem the input consists of the two sequences $(a[i])_{i=0}^{n-1}$ and $(b[i])_{i=0}^{n-1}$, the task is to compute their convolution $(c[i])_{i=0}^{n-1}$. A direct application of the definition of $(\min, +)$ -convolution yields a $\mathcal{O}(n^2)$ time algorithm to compute it. The bulk of the work of our algorithm consists of making a series of $(\min, +)$ -convolution steps. In fact, the running time of our algorithm can be stated as $\mathcal{O}(8^t \cdot t \cdot \log n \cdot \tau(t^2 n))$, where $\tau(n)$ is the running time of an algorithm computing the $(\min, +)$ -convolution of two sequences of length n . Therefore, there are two natural avenues for attempting to improve the algorithm of Theorem 1 to sub-quadratic. The first approach is to design a sub-quadratic algorithm for $(\min, +)$ -convolution, the second is to design an entirely different algorithm avoiding convolution altogether.

It turns out that the first approach is quite challenging, perhaps even impossible. Indeed, in the spirit of *fine-grained complexity* [23] analysis, Cygan et al. [12] identified a number of problems that admit algorithms with running time $\mathcal{O}(n^{2-\epsilon})$ if and only if $(\min, +)$ -CONVOLUTION does. With this background they conjecture that $(\min, +)$ -CONVOLUTION does *not* admit a $\mathcal{O}(n^{2-\epsilon})$ time algorithm.

Thus, if we want to improve the algorithm of Theorem 1 to a sub-quadratic algorithm without disproving the conjecture of Cygan et al. [12], we need to avoid $(\min, +)$ -convolution altogether. However, it turns out that $(\min, +)$ -convolution is unavoidable! In particular, we prove that a sub-quadratic algorithm for BISECTION on trees implies one for $(\min, +)$ -CONVOLUTION as well.

► **Theorem 2.** *If there exists an $\epsilon > 0$ such that BISECTION on weighted trees can be solved in time $\mathcal{O}(n^{2-\epsilon})$, then there exists $\delta > 0$ such that $(\min, +)$ -CONVOLUTION can be solved in $\mathcal{O}(n^{2-\delta})$ -time.*

Theorem 2 together with Theorem 1 (or rather its re-statement in terms of convolutions), puts BISECTION on weighted trees in Cygan et al. [12]’s class of problems equivalent to $(\min, +)$ -CONVOLUTION.

¹ Note that the Goldberg and Miller’s algorithm [16] is parallel, while ours is sequential.

In light of Theorem 2, the BISECTION problem on *unweighted* graphs² becomes a natural target. Our final contribution is a sub-quadratic algorithm for BISECTION on unweighted graphs of bounded treewidth. Our algorithm also works for the case when all weights are bounded by a constant W .

► **Theorem 3.** *There is an algorithm that, given an edge-weighted graph G , where all edge weights are integers between 1 and W , together with a tree decomposition of G of width t , computes a minimum weight bisection of G in time $\mathcal{O}(8^t \cdot (tW)^{\mathcal{O}(1)} \cdot n^{1.864} \log n)$.*

The key observation behind the algorithm of Theorem 3 is that the $(\min, +)$ -convolution steps in the algorithm of Theorem 1 are applied to sequences $(a[i])_{i=0}^{n-1}$ and $(b[i])_{i=0}^{n-1}$ where $a[i]$ and $b[i]$ are both essentially equal to the minimum possible sum of weights of the edges between the two sides A and B of a partition (A, B) of $V(G)$ with $|A| = i$. Bounded treewidth graphs have many vertices of small degree, and moving one vertex of small degree from A to B or vice versa changes the number of edges between A and B by at most its degree. Thus, $a[i]$ and $a[i + 1]$ cannot be too different. This allows us to use the faster algorithm for $(\min, +)$ -CONVOLUTION of Chan and Lewenstein [9] for “bounded difference” sequences.

Organization of the paper. We start by setting up the needed notation in Section 2. Section 3 is devoted to proving our algorithmic results - namely Theorems 1 and 3. Theorem 2 is proved in Section 4.

2 Preliminaries

2.1 The $(\min, +)$ -Convolution problem

For integer n , we let $[n] := \{0, 1, \dots, n\}$. Given a vector or a sequence $A \in \mathbb{Z}^n$ and an integer $i \in [n - 1]$, we denote by A_i the i -th coordinate of A .

► **Definition 4** ($(\min, +)$ -CONVOLUTION problem). *Given two sequences $(a[i])_{i=0}^{n-1}$ and $(b[i])_{i=0}^{n-1}$, compute a third sequence $(c[i])_{i=0}^{n-1}$, where $c[k] = \min_{i=0, \dots, k} (a[i] + b[k - i])$. Equivalently, we have $c[k] = \min_{i+j=k} (a[i] + b[j])$.*

In the $(\min, +)$ -CONVOLUTION problem, we sometimes require the target sequence to be computed all the way up to $2n - 2$, i.e., $(c[i])_{i=0}^{2n-2}$. In both cases, the problem is trivially solvable in $\mathcal{O}(n^2)$ time. Recent breakthroughs have shown that computing the $(\min, +)$ -CONVOLUTION for monotone non-decreasing sequences with integer values bounded by $\mathcal{O}(n)$ can be achieved in $\mathcal{O}(n^{1.864})$ deterministic time [9]. Moreover, we can relax these requirements [4] and simply require that the sequences have bounded differences, i.e., $|a[i] - a[i + 1]|, |b[i] - b[i + 1]| \in \mathcal{O}(1)$.

2.2 Graphs and the Bisection problem

We assume that each graph G is finite, simple, and undirected. We let $V(G)$ and $E(G)$ denote the vertex set and edge set of G , respectively. The *open neighborhood* of a vertex v is denoted by $N_G(v) = \{u \mid \{u, v\} \in E(G)\}$ and the *closed neighborhood* by $N_G[v] = N_G(v) \cup \{v\}$. For a set of vertices $S \subseteq V(G)$, we define $N_G(S) = \{v \notin S \mid \{u, v\} \in E(G), u \in S\}$ and $N_G[S] = N_G(S) \cup S$. The subgraph of G induced by S is denoted by $G[S]$, where $G[S]$ has vertex set S and edge set $\{\{u, v\} \in E(G) \mid u, v \in S\}$. We let $G - S = G[V(G) \setminus S]$.

² where all weights are 1

Given a graph G and two disjoint sets $A, B \subseteq V(G)$, we denote by $E(A, B)$ the subset of edges of G with one endpoint in A and the other endpoint in B . Given an edge-weighted graph G and a weight function $w : E(G) \rightarrow \mathbb{N}$ over the edges of G , a *bisection* of G is a partition of $V(G)$ into two disjoint sets $A, B \subseteq V(G)$ such that $||A| - |B|| \leq 1$ and the *weight of bisection* (A, B) is $\sum_{e \in E(A, B)} w(e)$. Formally, the BISECTION problem is defined as follows:

► **Definition 5** (BISECTION problem). *Given an edge-weighted graph G , find a bisection (A, B) of G of minimum weight.*

2.3 Treewidth and tree decompositions

► **Definition 6.** *A tree decomposition of a graph G is a pair $(\{X_i \mid i \in I\}, T = (I, F))$, where $\{X_i \mid i \in I\}$ is a collection of subsets of $V(G)$, $T = (I, F)$ is a rooted tree such that the following conditions hold:*

- $\bigcup_{i \in I} X_i = V(G)$;
- For all edges $\{u, v\} \in E(G)$, there exists $i \in I$ with $u, v \in X_i$;
- For every vertex $v \in V(G)$, the subgraph of T induced by $\{i \in I \mid v \in X_i\}$ is connected.

The width of a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is $\max_{i \in I} (|X_i| - 1)$. The treewidth of a graph G , $\text{tw}(G)$, is the minimum width over all possible tree decompositions of the graph. We call the vertices of the tree T nodes and the sets X_i bags. A graph of treewidth $\mathcal{O}(1)$ is called a bounded treewidth graph.

Given a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ of an n -vertex graph G of treewidth k , we can turn this decomposition in time in $\mathcal{O}(k^{\mathcal{O}(1)} \cdot n)$ into a *nice tree decomposition* with at most $\mathcal{O}(k|V(G)|)$ nodes, i.e., a decomposition of the same width and satisfying the following properties:

- The root bag as well as all leaf bags are empty;
- Every node of the tree decomposition is of one of four different types:
 - Leaf node: a node i with $X_i = \emptyset$ and no children;
 - Introduce node: a node i with exactly one child j such that $X_i = X_j \cup \{v\}$ for some vertex $v \in X_j$;
 - Forget node: a node i with exactly one child j such that $X_i = X_j \setminus \{v\}$ for some vertex $v \in X_j$;
 - Join node: a node i with two children j_1 and j_2 such that $X_i = X_{j_1} = X_{j_2}$.

► **Theorem 7** (Bodlaender et al. [2]). *There exists an algorithm, that given an n -vertex graph G and an integer k , in time $2^{\mathcal{O}(k)} n \log n$ either outputs that the treewidth of G is larger than k , or constructs a tree decomposition of G of width at most $3k + 4$.*

Combining Theorem 8 below with standard arguments (we refer the reader to [2] for more details), we arrive at Proposition 9, which is the form that will be required to obtain our algorithms.

► **Theorem 8** (Bodlaender and Hagerup [3]). *There is an algorithm that, given a tree decomposition of width k with $\mathcal{O}(n)$ nodes of a graph G , finds a rooted binary tree decomposition of G of width at most $3k + 2$ with depth $\mathcal{O}(\log n)$ in $\mathcal{O}(kn)$ -time.*

► **Proposition 9.** *There is an algorithm that, given an n -vertex graph G and a tree decomposition of G of width k , runs in $\mathcal{O}(kn)$ -time, and computes a nice tree decomposition of G of width $3k + 2$, height $\mathcal{O}(k \log n)$, and with $\mathcal{O}(kn)$ nodes.*

3 Algorithms for Bisection on Bounded Treewidth Graphs

We start by reviewing the $\mathcal{O}(2^{t+1} \cdot n^3)$ -time algorithm for solving the BISECTION problem on graphs of treewidth at most t by Jansen et al. [17]. The algorithm is a standard dynamic programming algorithm over a tree decomposition. Given a graph G together with its nice tree decomposition $(\{X_i | i \in I\}, T = (I, F))$ of width t the algorithm works as follows.

For each node $i \in I$, we let Y_i denote the set of all vertices in X_j , where either j is a descendant of i in T or $j = i$. The algorithm computes for each $i \in I$, an array mwp_i (which stands for minimum weight partition) containing $\mathcal{O}(2^t \cdot |Y_i|)$ entries. For each $\ell \in \{0, 1, \dots, |Y_i|\}$ and each $S \subseteq X_i$, the entry $\text{mwp}_i(\ell, S)$ is set to $\min_{S' \subseteq Y_i, |S'|=\ell, S' \cap X_i=S} (\sum_{e \in E(S', Y_i \setminus S')} w(e))$. That is, $\text{mwp}_i(\ell, S)$ is equal to the minimum possible weight of a bisection where S and $X_i \setminus S$ are in different parts of the bisection and the side including S is of cardinality exactly ℓ . When such a partition is not possible, we set $\text{mwp}_i(\ell, S)$ to ∞ .

We compute the entries of the array following the levels of the tree decomposition in a bottom-up manner as follows.

- Let i be a leaf in T . Note that $Y_i = X_i = \emptyset$. We set $\text{mwp}_i(0, \emptyset) = 0$.
- Let i be a forget node with one child j such that $X_i \subseteq X_j$. Then, for all $\ell \in \{0, 1, \dots, |Y_i|\}$ and $S \subseteq X_i$, we set

$$\text{mwp}_i(\ell, S) = \min_{S' \subseteq X_j, S' \cap X_i=S} (\text{mwp}_j(\ell, S')).$$

- Let i be an introduce node with one child j such that $X_j \cup \{v\} = X_i$ and $v \notin X_j$. Then, for all $\ell \in \{0, 1, \dots, |Y_i|\}$ and $S \subseteq X_i$, if $v \in S$ we set $\text{mwp}_i(\ell, S) = \text{mwp}_j(\ell - 1, S \setminus \{v\})$. Otherwise, we set

$$\text{mwp}_i(\ell, S) = \text{mwp}_j(\ell, S) + \sum_{e \in \{\{v, s\} | s \in S\}} w(e).$$

- Let i be a join node with two children j_1 and j_2 , where $X_i = X_{j_1} = X_{j_2}$. For all $\ell \in \{0, 1, \dots, |Y_i|\}$ and $S \subseteq X_i$, we set

$$\text{mwp}_i(\ell, S) = \min_{\ell_1 + \ell_2 = \ell, \ell_1, \ell_2 \geq |S|} \left(\text{mwp}_{j_1}(\ell_1, S) + \text{mwp}_{j_2}(\ell_2, S) - \sum_{e \in E(S, X_i \setminus S)} w(e) \right).$$

We omit the proof of correctness and refer the reader to [17] for more details. We focus here on the runtime analysis. Analyzing the above algorithm on the tree decomposition of width t and height $\mathcal{O}(t \log n)$, we obtain the following lemma.

► **Lemma 10.** *There is an algorithm that, given an edge-weighted graph G on n vertices and a nice tree decomposition of width t , height $\mathcal{O}(t \log n)$, and $\mathcal{O}(tn)$ nodes, computes a minimum weight bisection of G in time $\mathcal{O}(2^{t+1} \cdot t \cdot \log n \cdot \tau(t^2 n))$, where $\tau(|Y_i|)$ is the time required to compute the entries $\text{mwp}_i(\ell, S)$ for all $\ell \in [|Y_i|]$ and a fixed S in a join node.*

Proof. Let $(\{X_i | i \in I\}, T = (I, F))$ be the nice tree decomposition of G given as input. The time spent at each leaf node, introduce node, or forget node i is bounded by $\mathcal{O}(2^{t+1} \cdot |Y_i|)$. Moreover, by our assumption the time spent in each join node is $\mathcal{O}(2^{t+1} \tau(|Y_i|))$.

Now let us split the nodes of T into $r = \mathcal{O}(t \log n)$ levels L_0, \dots, L_r depending on the distance of the node from the root of T . We analyze the running time on each level separately. Clearly, the running time at level k is at most $\mathcal{O}(\sum_{i \in L_k} 2^{t+1} \tau(|Y_i|))$. Moreover, given $i, j \in L_k$ the nodes i and j cannot be descendants of each other. Therefore, from the

definition of a tree decomposition and Y_i and Y_j respectively, it follows that $Y_i \cap Y_j \subseteq X_i \cap X_j$. Hence, $\sum_{i \in L_k} |Y_i| \leq \sum_{i \in L_k} |X_i| + n \leq \sum_{i \in I} |X_i| + n \leq \mathcal{O}(t^2 n)$. Clearly $\tau(|Y_i|) = \Omega(|Y_i|)$ and it follows that $\mathcal{O}(\sum_{i \in L_k} 2^{t+1} \tau(|Y_i|)) \leq \mathcal{O}(2^{t+1} (\sum_{i \in L_k} \tau(|Y_i|))) \leq \mathcal{O}(2^{t+1} \tau(t^2 n))$. Combined with the fact that the height of the tree decomposition is $\mathcal{O}(t \log n)$, we get the claimed running time of $\mathcal{O}(2^{t+1} \cdot t \cdot \log n \cdot \tau(t^2 n))$. ◀

► **Lemma 11.** *Let i be a join node with children j_1 and j_2 , where $X_i = X_{j_1} = X_{j_2}$. There is an algorithm that, for a fixed $S \subseteq X_i$, computes all the entries $\text{mwp}_i(\ell, S)$, for all $\ell \in [|Y_i|]$, in time $\mathcal{O}(\tau(|Y_i|))$ if and only if there is an $\mathcal{O}(\tau(|Y_i|))$ time algorithm solving an instance of $(\min, +)$ -CONVOLUTION with two sequences $(a[p])_{p=0}^{|Y_i|}$ and $(b[p])_{p=0}^{|Y_i|}$, where $a[p] = \text{mwp}_{j_1}(p, S)$ for $p \in [|Y_{j_1}|]$ and $a[p] = \infty$ otherwise and $b[p] = \text{mwp}_{j_2}(p, S)$ for $p \in [|Y_{j_2}|]$ and $b[p] = \infty$ otherwise.*

Proof. Recall that

$$\text{mwp}_i(\ell, S) = \min_{\ell_1 + \ell_2 - |S| = \ell, \ell_1, \ell_2 \geq |S|} \left(\text{mwp}_{j_1}(\ell_1, S) + \text{mwp}_{j_2}(\ell_2, S) - \sum_{e \in E(S, X_i \setminus S)} w(e) \right).$$

Let $W = \sum_{e \in E(S, X_i \setminus S)} w(e)$. Note that for a fixed i and a fixed S , both $\sum_{e \in E(S, X_i \setminus S)} w(e)$ and $|S|$ are fixed. Hence,

$$\text{mwp}_i(\ell, S) = \min_{\ell_1 + \ell_2 - |S| = \ell, \ell_1, \ell_2 \geq |S|} (\text{mwp}_{j_1}(\ell_1, S) + \text{mwp}_{j_2}(\ell_2, S)) - W.$$

Let $(c[p])_{p=0}^{2|Y_i|-1}$ be the $(\min, +)$ -convolution of the sequences $(a[p])_{p=0}^{|Y_i|}$ and $(b[p])_{p=0}^{|Y_i|}$; that is $c[k] = \min_{q+r=k} (a[q] + b[r])$. Finally, we set $\text{mwp}_i(p, S) = c[p - |S|] - W$, for $p \in \{|S|, |S| + 1, \dots, |Y_i|\}$. All other entries are set to ∞ . ◀

Combining Lemmas 10 and 11 with Theorem 8 we conclude the proof of Theorem 1. We remark that if a tree decomposition is not given then we can compute it, using the algorithm of Theorem 7, at the cost of a worse dependence on t .

Proof of Theorem 1. We assume that $(\min, +)$ -CONVOLUTION can be solved in $\mathcal{O}(\tau(n))$ time. Using Proposition 9, we can compute in $\mathcal{O}(tn)$ time a nice tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ of G , such that the width of the decomposition is $3t + 2$, the height is $\mathcal{O}(t \log n)$, and the number of nodes of T is $\mathcal{O}(tn)$. Afterwards, we invoke the algorithm of Lemma 10 to compute the minimum weight bisection in time $\mathcal{O}(2^{3t+3} \cdot (3t + 2) \cdot \log n \cdot \tau((3t + 2)^2 n)) = \mathcal{O}(8^t \cdot t \cdot \log n \cdot \tau(t^2 n))$ using the $\mathcal{O}(\tau(|Y_i|))$ time algorithm to compute the $(\min, +)$ -convolution needed in the join nodes. Plugging in the naive $\mathcal{O}(n^2)$ time algorithm for $(\min, +)$ -CONVOLUTION gives $\tau(n) = \mathcal{O}(n^2)$, completing the proof. ◀

3.1 Bounded Edge Weights

We now turn our attention to the case when the maximum weight of every edge in the input graph is bounded by some constant W . We show that in this case, we can actually compute a minimum bisection of a bounded treewidth graph of size n in time $\mathcal{O}(8^t \cdot (tW)^{\mathcal{O}(1)} \cdot n^{1.864} \log n)$ or, equivalently, $\mathcal{O}(8^t \cdot (tW)^{\mathcal{O}(1)} \cdot n^{1.864+\epsilon})$, for $\epsilon > 0$.

► **Lemma 12.** *Let G be an edge-weighted graph with maximum weight of an edge W with a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ of width t . Then for every node $i \in I$, every $S \subseteq X_i$ and every $\ell \in \{|S|, \dots, |Y_i| - |X_i \setminus S| - 1\}$ it holds that $|\text{mwp}_i(\ell, S) - \text{mwp}_i(\ell + 1, S)| \leq (2t + 1) \cdot W$.*

Proof. It is easy to see that $\text{mwp}_i(\ell, S) = \text{mwp}_i(|Y_i| - \ell, X_i \setminus S)$. Hence, without loss of generality, we can assume that $\text{mwp}_i(\ell, S) \geq \text{mwp}_i(\ell + 1, S)$. Now let A be a set of size ℓ such that $S = A \cap X_i$ and $\text{mwp}_i(\ell, S) = \sum_{e \in E(A, \overline{A})} w(e)$. It is well-known that we can order the vertices of a graph G such that every vertex has at most $\text{tw}(G)$ neighbors earlier in the ordering [19]. Let us denote such an ordering σ and let v be the last vertex from $Y_i \setminus (A \cup X_i)$ in σ . Now $E(A \cup \{v\}, \overline{A \cup \{v\}}) = (E(A, \overline{A}) \setminus E(\{v\}, A)) \cup E(\{v\}, \overline{A \cup \{v\}})$. It follows that $\text{mwp}_i(\ell + 1, S) \leq \text{mwp}_i(\ell, S) + |E(\{v\}, \overline{A \cup \{v\}})| \cdot W$. By the choice of v , all the vertices in $\overline{A \cup \{v\}}$ are either earlier in σ than v or in X_i . Moreover, v has only at most $\text{tw}(G)$ many neighbors that are earlier in σ than v and there are at most $t + 1$ vertices in X_i , hence $|E(\{v\}, \overline{A \cup \{v\}})| \leq \text{tw}(G) + t + 1$. Since $\text{tw}(G) \leq t$, the lemma follows. ◀

Observe, that the bound of Lemma 12 is tight up to a multiplicative constant. As an example achieving difference $|\text{mwp}_i(\ell, S) - \text{mwp}_i(\ell + 1, S)| \leq (t + 1) \cdot W$ take $S = X_i$ and an instance where the edges in Y_i have all weight W and are precisely all the pairs with one endpoint in X_i and the other in $Y_i \setminus X_i$.

Lemma 12 tells us that the restriction of the sequences $(a[p])_{p=0}^{|Y_i|}$ and $(b[p])_{p=0}^{|Y_i|}$ for which we need to compute the $(\min, +)$ -CONVOLUTION in Lemma 11 to entries that are not ∞ has bounded difference. However, these two restricted sequences might not have the same length and it is not straightforward how to adapt the algorithm by Chan and Lewenstein [9]. To overcome this issue, we use a standard trick to change these sequences to monotone non-decreasing sequences with integer values bounded by $\mathcal{O}(n)$ and pad the shorter sequence by some large value. This trick is outlined by Chan and Lewenstein [9] but never formally stated, we repeat it here for completeness.

► **Theorem 13** ([9]). **MONOTONE** $(\min, +)$ -CONVOLUTION *with all entries in $\{0, \dots, nD\}$ can be solved in time $\mathcal{O}((nD)^{1.859})$ by a randomized algorithm, or in time $\mathcal{O}((nD)^{1.864})$ deterministically.*

We remark that Chan and Lewenstein [9] do not explicitly state the dependence on D . It is easy to see from their arguments that the dependence on D is at most $\mathcal{O}(D^{1.864})$, but we suspect that it is much better.

► **Lemma 14.** *Let n_1, n_2 be two integers such that $n_1 \leq n_2$ and let sequences $(a[p])_{p=0}^{n_1}$ and $(b[p])_{p=0}^{n_2}$ be two sequences with the difference bounded by a constant D and all entries in $\{0, \dots, n_2 D'\}$, for some constant D' . Then we can compute the sequence $(c[p])_{p=0}^{n_1+n_2}$ such that $c[k] = \min_{i+j=k} (a[i] + b[j])$ in time $\mathcal{O}((2n_2(D + D'))^{1.864})$.*

Proof. To compute $(c[p])_{p=0}^{n_1+n_2}$ we start by changing the sequences $(a[p])_{p=0}^{n_1}$ and $(b[p])_{p=0}^{n_2}$ to bounded monotone sequences $(a'[p])_{p=0}^{n_1}$ and $(b'[p])_{p=0}^{n_2}$ by adding $D \cdot i$ to $a'[i]$ and $b'[i]$, respectively. Note that $\min_{i+j=k} (a[i] + b[j]) = \min_{i+j=k} (a'[i] + b'[j]) - D \cdot k$. Now let $C = \max(a'[n_1], b'[n_2])$. Finally, we create sequence $(a''[p])_{p=0}^{n_2}$ by setting $a''[p] = a'[p]$ if $a'[p]$ is defined and $a''[p] = 2C + 1$ otherwise. It is easy to see that $\min_{i+j=k} (a'[i] + b'[j]) = \min_{i+j=k} (a''[i] + b'[j])$ for all $k \in \{0, \dots, n_1 + n_2\}$. Therefore, to compute the $(\min, +)$ -convolution of the sequences $(a[p])_{p=0}^{n_1}$ and $(b[p])_{p=0}^{n_2}$ it suffices to compute the $(\min, +)$ -convolution of the sequences $(a''[p])_{p=0}^{n_2}$ and $(b'[p])_{p=0}^{n_2}$, which are both monotone with integer entries between 0 and $C \leq 2(D \cdot n_2 + n_2 D') + 1$ and the proof follows due to Theorem 13. ◀

We are now in position to prove Theorem 3.

Proof of Theorem 3. Same as in the proof of Theorem 1, we start by using Proposition 9 to compute a nice tree decomposition $(\{X_i | i \in I\}, T = (I, F))$ of G , such that the width of the decomposition is $3t + 2$, the height is $\mathcal{O}(t \log n)$, and the number of nodes of T is $\mathcal{O}(tn)$.

Afterwards, we invoke the algorithm of Lemma 10 to compute the minimum weight bisection in time $\mathcal{O}(8^t \cdot t \cdot \log n \cdot \tau(t^2 n))$, where $\mathcal{O}(\tau(|Y_i|))$ is the time required to compute the entries $\text{mwp}_i(\ell, S)$ for all $\ell \in [|Y_i|]$ and a fixed S in a join node.

It remains to show that we can compute $\text{mwp}_i(\ell, S)$ for all $\ell \in [|Y_i|]$ and a fixed S in time $\mathcal{O}((tW)^{\mathcal{O}(1)} \cdot |Y_i|^{1.864})$. By Lemma 11, this is equivalent to solving an instance of $(\min, +)$ -convolution with two sequences $(a[p])_{p=0}^{|Y_i|}$ and $(b[p])_{p=0}^{|Y_i|}$, where $a[p] = \text{mwp}_{j_1}(p, S)$ for $p \in [|Y_{j_1}|]$ and $a[p] = \infty$ otherwise and $b[p] = \text{mwp}_{j_2}(p, S)$ for $p \in [|Y_{j_2}|]$ and $b[p] = \infty$ otherwise. Note that $\text{mwp}_{j_1}(\ell, S)$ ($\text{mwp}_{j_2}(\ell, S)$) is set to ∞ if $\ell < |S|$ or $\ell > |Y_{j_1}| - |X_{j_1} \setminus S|$ ($\ell > |Y_{j_2}| - |X_{j_2} \setminus S|$). Hence, from Lemma 12 it follows that if both $a[p]$ and $a[p+1]$ (respectively $b[p]$ and $b[p+1]$) are finite, then $|a[p+1] - a[p]|$ (respectively $|b[p+1] - b[p]|$) is bounded by $(2t+1) \cdot W$, where W is the maximum weight of an edge in G , and hence it is constant. To finish the proof, let $n_{j_1} = |Y_{j_1}| - |S| - |X_{j_1} \setminus S|$ and $n_{j_2} = |Y_{j_2}| - |S| - |X_{j_2} \setminus S|$ and let sequences $(a'[p])_{p=0}^{n_{j_1}}$ and $(b'[p])_{p=0}^{n_{j_2}}$ be such that $a'[p] = a[p+|S|]$ and $b'[p] = b[p+|S|]$. That is a' and b' are created from a and b by removing ∞ from the sequences. For all $k \in \{2|S|, \dots, n_{j_1} + n_{j_2} + 2|S|\}$ (that is whenever $\min_{i+j=k}(a[i] + b[j]) \neq \infty$) it holds that $\min_{i+j=k}(a[i] + b[j]) = \min_{i'+j'=k}(a'[i'] + b'[j'])$. Therefore, to compute the $(\min, +)$ -convolution of the sequences $(a[p])_0^{|Y_i|}$ and $(b[p])_0^{|Y_i|}$, it suffices to compute the sequence $(c'[p])_0^{n_{j_1} + n_{j_2}}$ such that $c'[k] = \min_{i+j=k}(a'[i] + b'[j])$. Clearly, due to Lemma 12, $(a'[p])_{p=0}^{n_{j_1}}$ and $(b'[p])_{p=0}^{n_{j_2}}$ have difference bounded by $(6t+5) \cdot W$. Moreover, let $n' = \max(n_{j_1}, n_{j_2})$, then it is easy to see that both $a[|S|]$ and $b[|S|]$ are at most $|S| \cdot n' \cdot W \leq (3t+3) \cdot n' \cdot W$ and hence the entries in $(a'[p])_{p=0}^{n_{j_1}}$ and $(b'[p])_{p=0}^{n_{j_2}}$ are all integers between 0 and $(3t+3) \cdot n' \cdot W + (6t+5) \cdot W \cdot n' = (9t+8) \cdot W \cdot n'$. Therefore, we can compute the sequence $(c'[p])_0^{n_{j_1} + n_{j_2}}$ in $\mathcal{O}(((30t+26) \cdot W \cdot n')^{1.864})$ by Lemma 14, finishing the proof. \blacktriangleleft

4 Tree Bisection is as Hard as $(\min, +)$ -Convolution

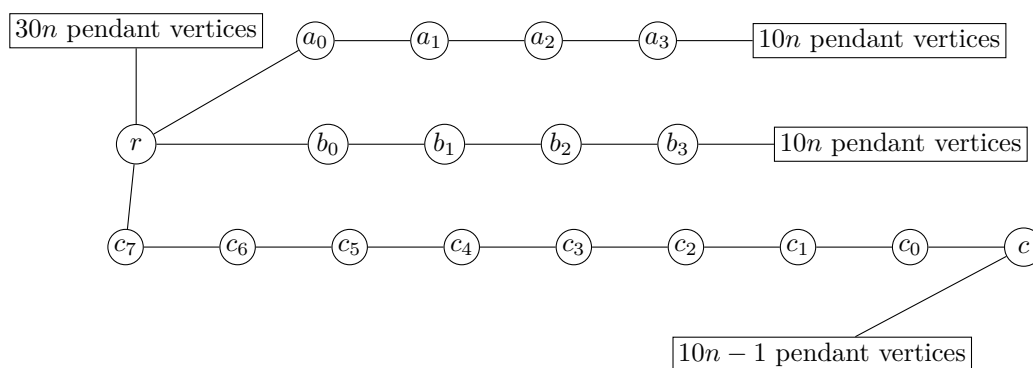
We complement Theorem 3 by showing that if the BISECTION problem can be solved in subquadratic time, i.e., in time $\mathcal{O}(n^{2-\epsilon})$ for $\epsilon > 0$, on weighted trees then the $(\min, +)$ -convolution problem can be solved in subquadratic time as well, i.e., in time $\mathcal{O}(n^{2-\delta})$ for $\delta > 0$. We follow a strategy similar to that of [1] used for proving a lower bound on the TREE SPARSITY problem.

► **Definition 15** (SUM3 problem). *Given three sequences $A, B, C \in \mathbb{Z}^n$, decide if the following statement is true: $\exists i, j : A_i + B_j + C_{i+j} \leq 0$.*

► **Theorem 16** ([1, 24]). *The $(\min, +)$ -CONVOLUTION problem can be solved in time $\mathcal{O}(n^{2-\epsilon})$, for $\epsilon > 0$, if and only if the SUM3 problem can be solved in $\mathcal{O}(n^{2-\delta})$ time, for $\delta > 0$.*

Hence, given Theorem 16, we prove the main theorem of this section by a reduction from SUM3 to the BISECTION problem on weighted trees. We start by describing the construction.

Let W be equal to 10 times the largest absolute value of an entry in A, B , and C . We create a root vertex r . Consider $A \in \mathbb{Z}^n$. We first construct a path $P_A = \{r, a_0, a_1, \dots, a_{n-1}\}$ of n vertices (excluding r) such that the weight of the i th edges is $W + A_i$, for $i = 0, 1, \dots, n-1$. Similarly, for $B \in \mathbb{Z}^n$, we construct a path $P_B = \{r, b_0, b_1, \dots, b_{n-1}\}$ of n vertices (excluding r) such that the weight of the i th edges is $W + B_i$, for $i = 0, 1, \dots, n-1$. We then create a new vertex c and a path $P_C = \{c, c_0, c_1, \dots, c_{n-1}, c_n, c_{n+1}, \dots, c_{2n-1}, r\}$ of $2n+1$ vertices such that the weight of the i th edges is $W + C_i$, for $i = 0, 1, \dots, n-1$ and the weight is nW otherwise ($i > n-1$). Finally, we attach $30n$ pendant vertices to r , $10n$ pendant vertices to



■ **Figure 1** The reduction from SUM3 (for $n = 4$) to the BISECTION problem on weighted trees.

a_{n-1} , $10n$ pendant vertices to b_{n-1} , and $10n - 1$ pendant vertices to c . The weight of each of those edges is nW . We let T denote the resulting tree (see Figure 1). Note that the total number of vertices in T is $60n + 4n = 64n$.

► **Lemma 17.** *Let $A, B, C \in \mathbb{Z}^n$ be an instance of SUM3 and let T be the corresponding instance of BISECTION. Then $\exists i, j : A_i + B_j + C_{i+j} \leq 0$ if and only if T has a bisection of weight less than or equal $3W$.*

Proof. Assume that $\exists i, j : A_i + B_j + C_{i+j} \leq 0$. We claim that T admits a bisection whose weight is at most $3W$. We pick one edge from each of the three paths P_A , P_B , and P_C . In particular, we pick the i -th edge from P_A , the j -th edge from P_B , and the k -th edge from P_C , where $k = i + j$. The total weight is therefore $3W + A_i + B_j + C_{i+j} \leq 3W$. The total number of vertices in the r -partition is $30n + i + j + 2n - k = 32n$ and the total number of vertices in the abc -partition is $30n + 2n + k - (i + j) = 32n$, as needed.

For the other direction, assume that T admits a bisection (X, Y) whose weight is at most $3W$. Notice, that from the choice of W and the construction, it follows that the weight of any at least four edges is at least $3W + \frac{6W}{10}$, and consequently $|E(X, Y)| \leq 3$. We claim that $E(X, Y)$ contains exactly three edges from T , each edge from a different path. Assume otherwise, i.e., that at least one path remains untouched. Then, the corresponding partition will contain at least $40n$ vertices which is greater than $32n$ vertices. Now, let $E(X, Y)$ contain the i -th edge from P_A , the j -th edge from P_B , and the k -th edge from P_C . It remains to show that $k = i + j$. The size of the partition containing r is $30n + i + j + 2n - k$. Since the number of vertices in T is $64n$ and both partitions must have equal size, we get $30n + i + j + 2n - k = 32n$ and therefore $i + j = k$, as needed. ◀

The construction, together with Proposition 16 and Lemma 17 conclude the proof of Theorem 2.

References

- 1 Arturs Backurs, Piotr Indyk, and Ludwig Schmidt. Better Approximations for Tree Sparsity in Nearly-linear Time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 2215–2229, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=3039686.3039831>.
- 2 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k n$ 5-Approximation Algorithm for Treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016. doi:10.1137/130947374.

- 3 Hans L. Bodlaender and Torben Hagerup. Parallel Algorithms with Optimal Speedup for Bounded Treewidth. *SIAM J. Comput.*, 27(6):1725–1746, December 1998. doi:10.1137/S0097539795289859.
- 4 Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly Subcubic Algorithms for Language Edit Distance and RNA-Folding via Fast Bounded-Difference Min-Plus Product. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 375–384. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.48.
- 5 Thang Nguyen Bui, Soma Chaudhuri, Frank Thomson Leighton, and Michael Sipser. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7(2):171–191, 1987.
- 6 Thang Nguyen Bui, C. Heigham, Curt Jones, and Frank Thomson Leighton. Improving the Performance of the Kernighan-Lin and Simulated Annealing Graph Bisection Algorithms. In Donald E. Thomas, editor, *Proceedings of the 26th ACM/IEEE Design Automation Conference, Las Vegas, Nevada, USA, June 25-29, 1989.*, pages 775–778. ACM Press, 1989. doi:10.1145/74382.74527.
- 7 Thang Nguyen Bui and Andrew Peck. Partitioning Planar Graphs. *SIAM J. Comput.*, 21(2):203–215, 1992.
- 8 Thang Nguyen Bui and Lisa C. Strite. An Ant System Algorithm For Graph Bisection. In *GECCO*, pages 43–51. Morgan Kaufmann, 2002.
- 9 Timothy M. Chan and Moshe Lewenstein. Clustered Integer 3SUM via Additive Combinatorics. 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 31–40. ACM, 2015. doi:10.1145/2746539.2746568.
- 10 Bruno Courcelle. The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Inf. Comput.*, 85(1):12–75, 1990.
- 11 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Minimum bisection is fixed parameter tractable. In *STOC*, pages 323–332. ACM, 2014.
- 12 Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Włodarczyk. On Problems Equivalent to (min, +)-Convolution. *ACM Trans. Algorithms*, 15(1):14:1–14:25, 2019. URL: <https://dl.acm.org/citation.cfm?id=3293465>, doi:10.1145/3293465.
- 13 Uriel Feige and Robert Krauthgamer. A Polylogarithmic Approximation of the Minimum Bisection. *SIAM J. Comput.*, 31(4):1090–1118, 2002.
- 14 Uriel Feige, Robert Krauthgamer, and Kobbi Nissim. Approximating the minimum bisection size (extended abstract). In *STOC*, pages 530–536, 2000.
- 15 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 16 M Goldberg and Z Miller. A parallel algorithm for bisection width in trees. *Computers & Mathematics with Applications*, 15(4):259–266, 1988.
- 17 Klaus Jansen, Marek Karpinski, Andrzej Lingas, and Eike Seidel. Polynomial Time Approximation Schemes for MAX-BISECTION on Planar and Geometric Graphs. In *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science, STACS '01*, pages 365–375, Berlin, Heidelberg, 2001. Springer-Verlag. URL: <http://dl.acm.org/citation.cfm?id=646515.759237>.
- 18 Subhash Khot and Nisheeth K. Vishnoi. The Unique Games Conjecture, Integrality Gap for Cut Problems and Embeddability of Negative-Type Metrics into ℓ_1 . *J. ACM*, 62(1):8:1–8:39, 2015.
- 19 Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994. doi:10.1007/BFb0045375.
- 20 Robert Malcolm Macgregor. On partitioning a graph: a theoretical and empirical study. Technical report, UC Berkeley, 1979.
- 21 Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of*

- Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 255–264. ACM, 2008. doi:10.1145/1374376.1374415.
- 22 René van Bevern, Andreas Emil Feldmann, Manuel Sorge, and Ondrej Suchý. On the Parameterized Complexity of Computing Graph Bisections. In *WG*, pages 76–87, 2013.
 - 23 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, 2018.
 - 24 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic Equivalences Between Path, Matrix, and Triangle Problems. *J. ACM*, 65(5):27:1–27:38, August 2018. doi:10.1145/3186893.

Oracle-Based Primal-Dual Algorithms for Packing and Covering Semidefinite Programs

Khaled Elbassioni

Khalifa University of Science and Technology, Masdar City Campus,
P.O. Box 54224, Abu Dhabi, UAE
khaled.elbassioni@ku.ac.ae

Kazuhisa Makino

Research Institute for Mathematical Sciences (RIMS), Kyoto University, Kyoto 606-8502, Japan
makino@kurims.kyoto-u.ac.jp

Abstract

Packing and covering semidefinite programs (SDPs) appear in natural relaxations of many combinatorial optimization problems as well as a number of other applications. Recently, several techniques were proposed, that utilize the particular structure of this class of problems, to obtain more efficient algorithms than those offered by general SDP solvers. For certain applications, such as those described in this paper, it may be required to deal with SDPs with *exponentially* or *infinitely* many constraints, which are accessible only via an *oracle*. In this paper, we give an efficient primal-dual algorithm to solve the problem in this case, which is an extension of a *logarithmic-potential* based algorithm of Grigoriadis, Khachiyan, Porkolab and Villavicencio (SIAM Journal of Optimization 41 (2001)) for packing/covering linear programs.

2012 ACM Subject Classification Mathematics of computing → Semidefinite programming; Theory of computation → Numeric approximation algorithms

Keywords and phrases Semidefinite programs, packing and covering, logarithmic potential, primal-dual algorithms, approximate solutions

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.43

Related Version A full version of the paper is available at <https://arxiv.org/abs/1809.09698>.

1 Introduction

1.1 Packing and Covering SDPs

We denote by \mathbb{S}^n the set of all $n \times n$ real symmetric matrices and by $\mathbb{S}_+^n \subseteq \mathbb{S}^n$ the set of all $n \times n$ positive semidefinite matrices. Consider the following pairs of *packing-covering* semidefinite programs (SDPs):

$$\begin{array}{l|l} \begin{array}{l} z_I^* = \max \quad C \bullet X \\ \text{s.t.} \quad A_i \bullet X \leq b_i, \forall i \in [m] \\ X \in \mathbb{R}^{n \times n}, X \succeq 0 \end{array} & \begin{array}{l} \text{(P-I)} \\ \\ \\ \end{array} \\ \hline \begin{array}{l} z_{II}^* = \min \quad C \bullet X \\ \text{s.t.} \quad A_i \bullet X \geq b_i, \forall i \in [m] \\ X \in \mathbb{R}^{n \times n}, X \succeq 0 \end{array} & \begin{array}{l} \text{(C-II)} \\ \\ \\ \end{array} \\ \hline \begin{array}{l} z_I^* = \min \quad b^T y \\ \text{s.t.} \quad \sum_{i=1}^m y_i A_i \succeq C \\ y \in \mathbb{R}^m, y \geq 0 \end{array} & \begin{array}{l} \text{(C-I)} \\ \\ \\ \end{array} \\ \hline \begin{array}{l} z_{II}^* = \max \quad b^T y \\ \text{s.t.} \quad \sum_{i=1}^m y_i A_i \preceq C \\ y \in \mathbb{R}^m, y \geq 0 \end{array} & \begin{array}{l} \text{(P-II)} \\ \\ \\ \end{array} \end{array}$$



© Khaled Elbassioni and Kazuhisa Makino;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 43; pp. 43:1–43:15



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

where $C, A_1, \dots, A_m \in \mathbb{S}_+^n$ are (non-zero) positive semidefinite matrices, and $b = (b_1, \dots, b_n)^T \in \mathbb{R}_+^n$ is a nonnegative vector. In the above, $C \bullet X := \text{Tr}(CX) = \sum_{i=1}^n \sum_{j=1}^n c_{ij}x_{ij}$, and “ \succeq ” is the *Löwner order* on matrices: $A \succeq B$ if and only if $A - B$ is positive semidefinite. This type of SDPs arise in many applications, see, e.g. [19, 18] and the references therein.

We will make the following assumption throughout the paper:

(A) $b_i > 0$ and hence $b_i = 1$ for all $i \in [m]$.

It is known that, under assumption (A), *strong duality* holds for problems (P-I)-(C-I) (resp., (P-II)-(C-II)).

Let $\epsilon \in (0, 1]$ be a given constant. We say that (X, y) is an ϵ -optimal primal-dual solution for (P-I)-(C-I) if (X, y) is a primal-dual feasible pair such that

$$C \bullet X \geq (1 - \epsilon)b^T y \geq (1 - \epsilon)z_I^*. \quad (1)$$

Similarly, we say that (X, y) is an ϵ -optimal primal-dual solution for (P-II)-(C-II) if (X, y) is a primal-dual feasible pair such that

$$C \bullet X \leq (1 + \epsilon)b^T y \leq (1 + \epsilon)z_{II}^*. \quad (2)$$

Since in this paper we allow the number of constraints m in (P-I) (resp., (C-II)) to be *exponentially* (or even infinitely) large, we will assume the availability of the following *oracle*:

Max(Y) (resp., **Min**(Y)): Given $Y \in \mathbb{S}_+^n$, find $i \in \text{argmax}_{i \in [m]} A_i \bullet Y$ (resp., $i \in \text{argmin}_{i \in [m]} A_i \bullet Y$).

Note that an *approximation oracle* computing the maximum (resp., minimum) above within a factor of $(1 - \epsilon)$ (resp., $(1 + \epsilon)$) is also sufficient for our purposes.

Our objective in this paper is to develop oracle-based *primal-dual* algorithms that find ϵ -optimal solutions for (P-I)-(C-I) and (P-II)-(C-II). An interesting property of our algorithms which distinguishes them from most previously known algorithms is that they produce solutions which are *sparse*, in the following sense: A primal-dual solution (X, y) to (C-I) (resp., (P-II)) is said to be η -sparse, if the size of $\text{supp}(y) := \{i \in [m] : y_i > 0\}$ is at most η . Two applications for SDP’s with infinite/exponential number of constraints are given in Section 3.

1.2 Main Result and Related Work

Problems (P-I)-(C-I) and (P-II)-(C-II) can be solved using general SDP solvers, such as interior-point methods: for instance, the barrier method (see, e.g., [29]) can compute a solution, within an *additive* error of ϵ from the optimal, in time $O(\sqrt{nm}(n^3 + mn^2 + m^2) \log \frac{1}{\epsilon})$ (see also [1, 34]). However, due to the special nature of (P-I)-(C-I) and (P-II)-(C-II), better algorithms can be obtained. Most of the improvements are obtained by using *first order methods* [4, 6, 7, 2, 13, 19, 20, 21, 22, 28, 30, 31], or second order methods [17, 18]. In general, we can classify these algorithms according to whether they:

- (I) are *width-independent*: the running time of the algorithm depends *polynomially* on the bit length of the input; for example, in the case of (P-I)-(C-I), the running time is $\text{poly}(n, m, \mathcal{L}, \log \tau, \frac{1}{\epsilon})$, where \mathcal{L} is the maximum bit length needed to represent any number in the input; on the other hand, the running time of a width-dependent algorithm will depend polynomially on a “width parameter” ρ , which is polynomial in \mathcal{L} and τ ;
- (II) are *parallel*: the algorithm takes $\text{polylog}(n, m, \mathcal{L}, \log \tau) \cdot \text{poly}(\frac{1}{\epsilon})$ time, on a $\text{poly}(n, m, \mathcal{L}, \log \tau, \frac{1}{\epsilon})$ number of processors;

- (III) *output sparse solutions*: the algorithm outputs an η -sparse solution to (C-I) (resp., (P-II)), for $\eta = \text{poly}(n, \log m, \mathcal{L}, \log \tau, \frac{1}{\epsilon})$ (resp., $\eta = \text{poly}(n, \log m, \mathcal{L}, \frac{1}{\epsilon})$), where τ is a parameter that bounds the trace of any optimal solution X (see Section 2 for details);
- (IV) *are oracle-based*: the only access of the algorithm to the matrices A_1, \dots, A_m is via the maximization/minimization oracle, and hence the running time is independent of m .

Table 1 gives a summary¹ of the most relevant results together with their classifications, according to the four criteria described above. We note that almost all these algorithms for packing/covering SDP's are generalizations of similar algorithms for packing/covering linear programs (LPs), and most of them are essentially based on an *exponential potential function* in the form of *scalar exponentiation*, e.g., [4, 22], or *matrix exponentiation* [6, 7, 2, 21, 19]. For instance, several of these results use the scalar or matrix versions of the *multiplicative weights updates* (MWU) method (see, e.g., [5]), which are extensions of similar methods for packing/covering LPs [14, 15, 35, 32].

In [16], a different type of algorithm was given for covering LPs (indeed, more generally, for a class of concave covering inequalities) based on a *logarithmic* potential function. In this paper, we show that this approach can be extended to provide oracle-based algorithms for both versions of packing and covering SDPs:

► **Theorem 1.** *For any $\epsilon > 0$, there is a randomized algorithm that, for any given instance of (P-I)-(C-I), outputs an $O(n\mathcal{L} \log(n\tau) + \frac{n}{\epsilon^2})$ -sparse $O(\epsilon)$ -optimal primal-dual pair in time² $\tilde{O}(\frac{n^{\omega+1}\mathcal{L} \log \tau}{\epsilon^{2.5}} + \frac{n\mathcal{L}\mathcal{T} \log \tau}{\epsilon^2})$, where \mathcal{T} is the time taken by a single call to the oracle $\text{Max}(\cdot)$ and ω is the exponent of matrix multiplication.*

► **Theorem 2.** *For any $\epsilon > 0$, there is a randomized algorithm that, for any given instance of (P-II)-(C-II), outputs an $O(n\mathcal{L} \log n + \frac{n}{\epsilon^2})$ -sparse $O(\epsilon)$ -optimal primal-dual pair in time $\tilde{O}(\frac{n^{\omega+1}\mathcal{L} \log n}{\epsilon^{2.5}} + \frac{n\mathcal{L}\mathcal{T}}{\epsilon^2})$, where \mathcal{T} is the time taken by a single call to the oracle $\text{Min}(\cdot)$.*

As we can see from the table, among all the algorithms listed, the logarithmic-potential algorithm, presented in this paper, is the only one that produces sparse solutions, in the sense described above. Moreover, the only known other oracle-based algorithm (matrix Matrix MWU [7]) is *not* width-independent. It can also be shown (see [12]) that a modified version of the matrix exponential MWU algorithm [6] can yield sparse solutions for (P-II)-(C-II). However, the overall running time of this matrix MWU algorithm is larger by a factor of (roughly) $\Omega(n^{3-\omega})$ than that of the logarithmic-potential algorithm. Moreover, we were not able to extend the matrix MWU algorithm to solve (P-I)-(C-I) (in particular, it seems tricky to bound the number of iterations).

A work that is also related to ours is the sparsification of graph Laplacians [8] and positive semidefinite sums [33]. Given matrices $A_1, \dots, A_m \in \mathbb{S}_+^n$ and $\epsilon > 0$, it was shown in [33] that one can find, in $O(\frac{n}{\epsilon^2}(n^\omega + \mathcal{T}))$ time, a vector $y \in \mathbb{R}_+^m$ with support size $O(\frac{n}{\epsilon^2})$, such that $B \preceq \sum_i y_i A_i \preceq (1 + \epsilon)B$, where $B := \sum_i A_i$ and \mathcal{T} is the time taken by a single call to the minimization oracle $\text{Min}(Y)$ (for a not necessarily positive semidefinite matrix Y). An immediate corollary is that, given an ϵ -optimal solution y for (C-I) (resp., (P-I)), one can find in $O(\frac{n}{\epsilon^2}(n^\omega + \mathcal{T}))$ time an $O(\epsilon)$ -optimal solution y' with support size $O(\frac{n}{\epsilon^2})$. Interestingly, the algorithm in [33] (which is an extension for the rank-one version in [8]) uses the *barrier potential function* $\Phi'(x, F) := \text{Tr}((H - xI)^{-1})$ (resp., $\Phi'(x, H) := \text{Tr}((xI - H)^{-1})$), while in

¹ We provide rough estimates of the bounds, as some of them are not explicitly stated in the corresponding paper in terms of the parameters we consider here.

² $\tilde{O}(\cdot)$ hides polylogarithmic factors in n and $\frac{1}{\epsilon}$.

■ **Table 1** Different Algorithms for Packing/covering SDPs.

Paper	Problem	Technique	Most Expensive Operation	# Iterations	Width-indep.	Parallel	Sparse	Oracle-based
[4, 22]	(P-I) (C-II)	MWU	max / min eigenvalue of a PSD matrix $\tilde{O}(\frac{n^2}{\epsilon})$	$O(\frac{\rho \log m}{\epsilon})$	No	No	No [*]	No
[7]	(P-I) (C-II)	Matrix MWU	Matrix exponentiation $O(n^3)$	$O(\frac{\rho^2 \epsilon^2 \log n}{\epsilon^2 (z_1^*)^2})$	No	No	No [*]	Yes
[17, 19]	(P-I)	Nesterov's smoothing technique [27, 28]	Matrix exponentiation $O(n^3)$	$O(\frac{\tau \log m}{\epsilon})$	No	No	No	No
[18]	(C-II)	Nesterov's smoothing technique [27, 28]	min eigenvalue of a non PSD matrix $O(n^3)$	$O(\frac{\rho^2 \log(nm)}{\epsilon})$	No	No	No	No
[20]	(P-I)& (C-II)	MWU technique [27, 28]	eigenvalue decomposition $O(n^3)$	$O(\frac{\log^{13} n \log m}{\epsilon^{13}})$	Yes	Yes	No	No
[30, 31]	(P-II)& (C-II)	Matrix MWU	Matrix exponentiation $O(n^3)$	$O(\frac{\log^3 m}{\epsilon^3})$	Yes	Yes	No	No
[2]	(P-I)& (C-II)	Gradient Descent + Mirror Descent	Matrix exponentiation $O(n^3)$	$O(\frac{\log^2(mn) \log \frac{1}{\epsilon}}{\epsilon^2})$	Yes	Yes	No	No
[12]	(P-II)& (C-II)	Matrix MWU	Matrix exponentiation $O(n^3)$	$O(\frac{n \log n}{\epsilon^2})$	Yes	No	Yes	Yes
This paper	(P-II) & (C-II) (P-II) & (C-II)	Logarithmic potential [16]	Matrix inversion $O(n^w)$	$O(n \log(n\mathcal{L}\tau) + \frac{n}{\epsilon^2})$ $O(n \log(n/\epsilon) + \frac{n}{\epsilon^2})$	Yes	No	Yes	Yes

* In fact, these algorithms find sparse solutions, in the sense that the dependence of the size of the support of the dual solution on m is at most logarithmic; however, the dependence of the size of the support on the bit length \mathcal{L} is not polynomial.

our algorithms (generalizing the potential function in [16]) we use the logarithmic potential function $\Phi(x, H) = \ln x + \frac{\epsilon}{n} \ln \det(H - xI) = \ln x - \frac{\epsilon}{n} \int_x \Phi'(x, H) dx$ (resp., $\Phi(x, H) = \ln x - \frac{\epsilon}{n} \ln \det(xI - H) = \ln x - \frac{\epsilon}{n} \int_x \Phi'(x, H) dx$). Sparsification algorithms with better running times were recently obtained in [3, 24]. Since the sparse solutions produced by our algorithms may have support size slightly more (by polylogarithmic factors) than $O(\frac{n}{\epsilon^2})$, we may use, in a post-processing step, the sparsification algorithms, mentioned above, to convert the solutions obtained by Theorems 1 and 2 to ones with support size $O(\frac{n}{\epsilon^2})$, without increasing the overall asymptotic running time.

In Section 4, we give an outline of the algorithm and sketch the proof of Theorem 1; the proof of Theorem 2 is similar. To motivate our algorithms, in Section 3, we give two applications that require finding sparse solutions for a packing/covering SDP ³.

2 Reduction to Normalized Form

When $C = I = I_n$, the identity matrix in $\mathbb{R}^{n \times n}$ and $b = \mathbf{1}$, the vector of all ones in \mathbb{R}^m , we say that the packing-covering SDPs are in *normalized* form:

$$\begin{array}{l}
 z_i^* = \max \quad I \bullet X \quad \quad \quad \text{(N-P-I)} \\
 \text{s.t.} \quad A_i \bullet X \leq 1, \forall i \in [m] \\
 \quad \quad X \in \mathbb{R}^{n \times n}, X \succeq 0
 \end{array}
 \quad \left| \quad \quad \quad
 \begin{array}{l}
 z_i^* = \min \quad \mathbf{1}^T y \quad \quad \quad \text{(N-C-I)} \\
 \text{s.t.} \quad \sum_{i=1}^m y_i A_i \succeq I \\
 \quad \quad y \in \mathbb{R}^m, y \geq 0.
 \end{array}$$

³ As pointed out by an anonymous reviewer, solution-sparsity and oracle-access to the input can also be thought of as a way of reducing the *space requirement* of the algorithm, see [23].

$$\begin{array}{l}
z_{II}^* = \min \quad I \bullet X \quad (\text{N-C-II}) \\
\text{s.t.} \quad A_i \bullet X \geq 1, \forall i \in [m] \\
\quad X \in \mathbb{R}^{n \times n}, X \succeq 0
\end{array}
\left| \begin{array}{l}
z_{II}^* = \max \quad \mathbf{1}^T y \quad (\text{N-P-II}) \\
\text{s.t.} \quad \sum_{i=1}^m y_i A_i \preceq I \\
\quad y \in \mathbb{R}^m, y \geq 0.
\end{array} \right.$$

It can be shown⁴ that, at the loss of a factor of $(1 + \epsilon)$ in the objective, any pair of packing-covering SDPs of the form (P-I)-(C-I) can be brought in $O(n^3)$, increasing the oracle time only by $O(n^\omega)$, where ω is the exponent of matrix multiplication, to the normalized form (N-P-I)-(N-C-I), under the following assumption:

(B-I) There exist r matrices, say A_1, \dots, A_r , such that $\bar{A} := \sum_{i=1}^r A_i \succ 0$. In particular, $\text{Tr}(X) \leq \tau := \frac{r}{\lambda_{\min}(\bar{A})}$ for any optimal solution X for (P-I).

Similarly, one can show (see [12, 20]) that, at the loss of a factor of $(1 + \epsilon)$ in the objective, any pair of packing-covering SDPs of the form (P-II)-(C-II) can be brought in $O(n^3)$ time, increasing the oracle time only by $O(n^\omega)$, to the normalized form (N-P-II)-(N-C-II). Moreover, we may assume in this normalized form that

(B-II) $\lambda_{\min}(A_i) = \Omega\left(\frac{\epsilon}{n} \cdot \min_{i'} \lambda_{\max}(A_{i'})\right)$ for all $i \in [m]$,

where, for a positive semidefinite matrix $B \in \mathbb{S}_+^{n \times n}$, we denote by $\{\lambda_j(B) : j = 1, \dots, n\}$ the eigenvalues of B , and by $\lambda_{\min}(B)$ and $\lambda_{\max}(B)$ the minimum and maximum eigenvalues of B , respectively. With an additional $O(mn^2)$ time, we may also assume that:

(B-II') $\frac{\lambda_{\max}(A_i)}{\lambda_{\min}(A_i)} = O\left(\frac{n^2}{\epsilon^2}\right)$ for all $i \in [m]$.

Thus, from now on we focus on the normalized problems.

3 Applications

3.1 Robust Packing and Covering SDPs

Consider a packing-covering pair of the form (P-I)-(C-I) or (P-II)-(C-II). In the framework of *robust optimization* (see, e.g. [9, 10]), we assume that each constraint matrix A_i is not known exactly; instead, it is given by a convex uncertainty set $\mathcal{A}_i \subseteq \mathbb{S}_+^n$. It is required to find a (near)-optimal solution for the packing-covering pair under the *worst-case* choice $A_i \in \mathcal{A}_i$ of the constraints in each uncertainty set. A typical example of a *convex* uncertainty set is given by an *affine perturbation* around a nominal matrix $A_i^0 \in \mathbb{S}_+^n$:

$$\mathcal{A}_i = \left\{ A_i := A_i^0 + \sum_{r=1}^k \delta_r A_i^r : \delta = (\delta_1, \dots, \delta_k) \in \mathcal{D} \right\}, \quad (3)$$

where $A_i^1, \dots, A_i^k \in \mathbb{S}_+^n$, and $\mathcal{D} \subseteq \mathbb{R}_+^k$ can take, for example, one of the following forms:

- *Ellipsoidal uncertainty*: $\mathcal{D} = E(\delta_0, D) := \{\delta \in \mathbb{R}_+^k : (\delta - \delta_0)^T D^{-1} (\delta - \delta_0) \leq 1\}$, for given positive definite matrix $D \in \mathbb{S}_+^k$ and vector $\delta_0 \in \mathbb{R}_+^k$ such that $E(\delta_0, D) \subseteq \mathbb{R}_+^k$;
- *Polyhedral uncertainty*: $\mathcal{D} := \{\delta \in \mathbb{R}_+^k : D\delta \leq w\}$, for given matrix $D \in \mathbb{R}^{h \times k}$ and vector $w \in \mathbb{R}^h$.

⁴ In fact, unlike previously known reductions, the reduction we give in [12] is simpler to compute, as it is based on the *LDL-decompositions* rather than the *eigenvalue decompositions* of the input matrices.

Without loss of generality, we consider the robust version of (N-P-I)-(N-C-I), where A_i , for $i \in [m]$, belongs to a convex uncertainty set \mathcal{A}_i . Then the robust optimization problem and its dual can be written as follows:

$$\begin{array}{l|l}
 \begin{array}{l}
 z_P^* = \max \quad I \bullet X \\
 \text{s.t. } \quad A_i \bullet X \leq 1, \quad \forall A_i \in \mathcal{A}_i \quad \forall i \in [m] \\
 X \in \mathbb{R}^{n \times n}, \quad X \succeq 0
 \end{array} & \text{(R-P-I)} \\
 \hline
 \begin{array}{l}
 z_D^* = \inf \quad \sum_{i=1}^m \int_{\mathcal{A}_i} y_{A_i}^i dA_i \\
 \text{s.t. } \quad \sum_{i=1}^m \int_{\mathcal{A}_i} y_{A_i}^i A_i dA_i \succeq I \\
 y^i \text{ is a discrete measure on } \mathcal{A}_i, \quad \forall i \in [m].
 \end{array} & \text{(R-C-I)}
 \end{array}$$

As before, we assume (B-I), where $A_1, \dots, A_r \in \bigcup_{i \in [m]} \mathcal{A}_i$. We call a pair of solutions (X, y) to be ϵ -optimal for (R-P-I)-(R-C-I), if

$$z_P^* \geq I \bullet X \geq (1 - \epsilon) \sum_{i=1}^m \int_{\mathcal{A}_i} y_{A_i}^i dA_i \geq (1 - \epsilon) z_D^*.$$

Note that the number of constraints in (R-P-I) is *infinite* and hence any algorithm that solves the problem would have to be oracle-based. The Ellipsoid method is one such algorithm; a more efficient procedure is given by the following corollary of Theorem 1.

► **Theorem 3.** *For any $\epsilon > 0$, there is a randomized algorithm that outputs an $O(\epsilon)$ -optimal primal-dual pair for (R-P-I)-(R-C-I) in time $\tilde{O}\left(\frac{n^{\omega+1} \log \psi}{\epsilon^{2.5}} + \frac{nT \log \psi}{\epsilon^2}\right)$, where $\psi := \frac{r \cdot \max_{i \in [m], A_i \in \mathcal{A}_i} \lambda_{\max}(A_i)}{\lambda_{\min}(A)}$ and \mathcal{T} is the time to compute, for a given $Y \in \mathbb{S}_+^n$, a pair (i, A_i) such that*

$$(i, A_i) \in \operatorname{argmax}_{i \in [m], A_i \in \mathcal{A}_i} A_i \bullet Y. \quad (4)$$

Note that (4) amounts to solving a linear optimization problem over a convex set. Moreover, for simple uncertainty sets, such as boxes or ellipsoids, such computation can be done very efficiently.

3.2 Carr-Vempala-Type Decomposition

Consider a maximization (resp., minimization) problem over a discrete set $\mathcal{S} \subseteq \mathbb{Z}^n$ and a corresponding SDP-relaxation over $\mathcal{Q} \subseteq \mathbb{S}_+^n$:

$$\begin{array}{l|l}
 \begin{array}{l}
 z_{CO}^* = \left\{ \begin{array}{l} \max \\ \min \end{array} \right\} \quad C \bullet qq^T \\
 q \in \mathcal{S}
 \end{array} & \text{(DOP)} \\
 \hline
 \begin{array}{l}
 z_{SDP}^* = \left\{ \begin{array}{l} \max \\ \min \end{array} \right\} \quad C \bullet Q \\
 Q \in \mathcal{Q},
 \end{array} & \text{(SDP-RLX)}
 \end{array}$$

where $C \in \mathbb{S}_+^n$.

► **Definition 4.** *For $\alpha \in (0, 1]$ (resp., $\alpha \geq 1$), an α -integrality gap verifier \mathcal{A} for (SDP-RLX) is a polytime algorithm that, given any $C \in \mathbb{S}_+^n$ and any $Q \in \mathcal{Q}$ returns a $q \in \mathcal{S}$ such that $C \bullet qq^T \geq \alpha C \bullet Q$ (resp., $C \bullet qq^T \leq \alpha C \bullet Q$).*

For instance, if $\mathcal{S} = \{-1, 1\}^n$ and $\mathcal{Q} = \{X \in \mathbb{S}_+^n : X_{ii} = 1 \quad \forall i \in [n]\}$, then a $\frac{2}{\pi}$ -integrality gap verifier for the maximization version of (SDP-RLX) is known [26].

Carr and Vempala [11] gave a decomposition theorem that allows one to use an α -integrality gap verifier for a given LP-relaxation of a combinatorial maximization (resp., minimization) problem, to decompose a given fractional solution to the LP into a convex combination of integer solutions that is dominated by (resp., dominates) α times the fractional solution. Here we derive a similar result for SDP relaxations:

► **Theorem 5.** *Consider a combinatorial maximization (resp., minimization) problem (DOP) and its SDP relaxation (SDP-RLX), admitting an α -integrality gap verifier \mathcal{A} . Assume the set \mathcal{S} is full-dimensional and let $\epsilon > 0$ be a given constant. Then there is a polytime algorithm that, for any given $Q \in \mathcal{Q}$, finds a set $\mathcal{X} \subseteq \mathcal{S}$ of size $|\mathcal{X}| = O(\frac{n^3}{\epsilon^2} \log(nW))$ (resp., of size $|\mathcal{X}| = O(n \log \frac{n}{\epsilon} + \frac{n}{\epsilon^2})$), where $W := \max_{q \in \mathcal{S}, i \in [n]} |q_i|$, and a set of convex multipliers $\{\lambda_q \in \mathbb{R}_+ : q \in \mathcal{X}\}$, $\sum_{q \in \mathcal{X}} \lambda_q = 1$, such that*

$$(1 - O(\epsilon))\alpha Q \preceq \sum_{q \in \mathcal{X}} \lambda_q qq^T \quad (\text{resp.}, (1 + O(\epsilon))\alpha Q \succeq \sum_{q \in \mathcal{X}} \lambda_q qq^T). \quad (5)$$

The proof of Theorem 5 is obtained by considering the following pairs of packing and covering SDPs (of types I and II, respectively):

$z_I^* = \min \sum_{q \in \mathcal{S}} \lambda_q \quad (\text{CVX-I})$ <p>s.t. $\sum_{q \in \mathcal{S}} \lambda_q qq^T \succeq \alpha Q \quad (6)$</p> $\sum_{q \in \mathcal{S}} \lambda_q \geq 1 \quad (7)$ $\lambda \in \mathbb{R}^{\mathcal{S}}, \lambda \geq 0$		$z_I^* = \max \alpha Q \bullet Y + u \quad (\text{CVX-dual-I})$ <p>s.t. $qq^T \bullet Y + u \leq 1, \forall q \in \mathcal{S} \quad (8)$</p> $Y \in \mathbb{S}_+^n, u \geq 0.$
$z_{II}^* = \max \sum_{q \in \mathcal{S}} \lambda_q \quad (\text{CVX-II})$ <p>s.t. $\sum_{q \in \mathcal{S}} \lambda_q qq^T \preceq \alpha Q \quad (9)$</p> $\sum_{q \in \mathcal{S}} \lambda_q \leq 1 \quad (10)$ $\lambda \in \mathbb{R}^{\mathcal{S}}, \lambda \geq 0$		$z_{II}^* = \min \alpha Q \bullet Y + u \quad (\text{CVX-dual-II})$ <p>s.t. $qq^T \bullet Y + u \geq 1, \forall q \in \mathcal{S} \quad (11)$</p> $Y \in \mathbb{S}_+^n, u \geq 0.$

It can be shown, using the fact that the SDP relaxation admits an α -integrality gap verifier, that $z_I^* = z_{II}^* = 1$, and that the two primal-dual pairs can be solved in polynomial time using the Ellipsoid method. A more efficient (but approximate version) can be obtained using the algorithms of Theorems 1 and 2. Note that, once we have a set \mathcal{X} as in Theorem 5, its support can be reduced to $O(\frac{n^2}{\epsilon})$ using the sparsification techniques of [8, 33].

4 A Logarithmic Potential Algorithm for (P-I)-(C-I)

In this section we give an algorithm for finding a sparse $O(\epsilon)$ -optimal primal-dual solution for (N-P-I)-(N-C-I). Since the algorithm updates only one component of the dual solution in each iteration, it follows that the number of positive components of the dual solution when the algorithm terminates is exactly equal to the number of iterations; from this sparsity follows.

4.1 High-level Idea of the Algorithm

The idea of the algorithm is quite intuitive. It can be easily seen that problem (N-C-I) is equivalent to finding a convex combination of the A_i 's that maximizes the minimum eigenvalue, that is, $\max_{y \in \mathbb{R}_+^m: \mathbf{1}^T y = 1} \lambda_{\min}(F(y))$, where $F(y) := \sum_{i=1}^m y_i A_i$, and $\mathbf{1}$ is the m -dimensional vector of all ones. Since $\lambda_{\min}(F(y))$ is not a *smooth* function in y , it is more convenient to work with a smooth approximation of it, which is obtained by maximizing (over x) a *logarithmic potential function* $\Phi(x, F(y))$ that captures the constraints that each eigenvalue of $F(y)$ is at least x . The unique maximizer $x = \theta^*$ of $\Phi(x, F(y))$ defines a set of “weights” (these are the eigenvalues of the primal matrix X computed in line 6 of the algorithm) such that the weighted average of the $\lambda_j(F(y))$'s is a very close approximation of $\lambda_{\min}(F(y))$. Thus, to maximize this average (which is exactly $X \bullet F(y)$), we obtain a direction (line 7) along which y is modified with an appropriate step size (line 10). For numbers $x \in \mathbb{R}_+$ and $\delta \in (0, 1)$, a δ -(lower) approximation x_δ of x is a number such that $(1 - \delta)x \leq x_\delta < x$. For $i \in [m]$, $\mathbf{1}_i$ denotes the i th unit vector of dimension m .

The algorithm is shown as Algorithm 1. The main while-loop (step 4) is embedded within a sequence of scaling phases, in which each phase starts from the vector $y(t)$ computed in the previous phase and uses double the accuracy. The algorithm stops when the scaled accuracy ε_s drops below the desired accuracy $\epsilon \in (0, 1/2)$. When referring to an arbitrary iteration of the algorithm, we assume it is iteration t in phase s .

4.2 Analysis

4.2.1 High-level Idea of the Analysis

The analysis is based on a matrix generalization of the scalar arguments given in [16] (as is the case for all algorithms for SDP's, which are driven from their LP counterparts; see, e.g., [1]). Besides the technical details, the algorithm also requires estimating the minimum eigenvalue of the dual matrix $F(y(t))$, which is done using Lanczos' algorithm (see Section 4.2.4 for details).

The proof of ϵ -optimality follows easily from the stopping condition in line 4 of the algorithm, the definition of the “approximation error” ν in line 8, and the fact that $X \bullet F(y)$ is a very close approximation of $\lambda_{\min}(F(y(t)))$. The main part of the proof is to bound the number of iterations in the inner while-loop (line 4). This is done by using a *potential function argument*: we define the potential function $\Phi(t) := \Phi(\theta^*(t), F(y(t)))$ and show in Claim 23 that, in each iteration, the choice of the step size in line 9 guarantees that $\Phi(t)$ is increased substantially; on the other hand, by Claim 24, the potential difference cannot be very large, and the two claims together imply that we cannot have many iterations.

4.2.2 Some Preliminaries

Up to Claim 26, we fix a particular iteration s of the outer while-loop in the algorithm. For simplicity in the following, we will sometimes write $F := F(y(t))$, $\theta := \theta(t)$, $\theta^* := \theta^*(t)$, $X := X(t)$, $\hat{F} := A_{i(t)}$, $\tau := \tau(t + 1)$, $\nu := \nu(t + 1)$, $F' := F(y(t + 1))$, and $\theta' := \theta(t + 1)$, when the meaning is clear from the context. For $H \succ 0$ and $x \in (0, \lambda_{\min}(H))$, define the logarithmic potential function [16, 29]:

$$\Phi(x, H) = \ln x + \frac{\varepsilon_s}{n} \ln \det (H - xI). \quad (12)$$

Note that the term $\ln \det (H - xI)$ forces the value of x to stay away from the “boundary” $\lambda_{\min}(H)$, while the term $\ln x$ pushes x towards that boundary; hence, one would expect the maximizer of $\Phi(x, H)$ to be a good approximation of $\lambda_{\min}(H)$ (see Claim 8).

■ **Algorithm 1** Logarithmic-potential Algorithm for (P-I)-(C-I).

```

1  $s \leftarrow 0$ ;  $\varepsilon_0 \leftarrow \frac{1}{2}$ ;  $t \leftarrow 0$ ;  $\nu(0) \leftarrow 1$ ;  $y(0) \leftarrow \frac{1}{r} \sum_{i=1}^r \mathbf{1}_i$ 
2 while  $\varepsilon_s > \epsilon$  do
3    $\delta_s \leftarrow \frac{\varepsilon_s^3}{32n}$ 
4   while  $\nu(t) > \varepsilon_s$  do
5      $\theta(t) \leftarrow \theta^*(t)_{\delta_s}$ , where  $\theta^*(t)$  is the smallest positive number root of the
       equation  $\frac{\varepsilon_s \theta}{n} \text{Tr}(F(y(t)) - \theta I)^{-1} = 1$ 
6      $X(t) \leftarrow \frac{\varepsilon_s \theta(t)}{n} (F(y(t)) - \theta(t)I)^{-1}$  /* Set the primal solution */
7      $i(t) \leftarrow \text{argmax}_i A_i \bullet X(t)$  /* Call the maximization oracle */
8      $\nu(t+1) \leftarrow \frac{X(t) \bullet A_{i(t)} - X(t) \bullet F(y(t))}{X(t) \bullet A_{i(t)} + X(t) \bullet F(y(t))}$  /* Compute the error */
9      $\tau(t+1) \leftarrow \frac{\varepsilon_s \theta(t) \nu(t+1)}{4n(X(t) \bullet A_{i(t)} + X(t) \bullet F(y(t)))}$  /* Compute the step size */
10     $y(t+1) \leftarrow (1 - \tau(t+1))y(t) + \tau(t+1)\mathbf{1}_{i(t)}$  /* Update the dual solution */
11     $t \leftarrow t + 1$ 
12  end
13   $\varepsilon_{s+1} \leftarrow \frac{\varepsilon_s}{2}$ 
14   $s \leftarrow s + 1$ 
15 end
16  $\hat{X} \leftarrow \frac{(1-\varepsilon_{s-1})X(t-1)}{(1+\varepsilon_{s-1})^2\theta(t-1)}$ ;  $\hat{y} \leftarrow \frac{y(t-1)}{\theta(t-1)}$  /* Scale primal and dual to retain feasibility */
17 return  $(\hat{X}, \hat{y}, t)$ 

```

▷ **Claim 6.** If $F(y(t)) \succ 0$, then $\theta^*(t) = \text{argmax}_{0 < x < \lambda_{\min}(F)} \Phi(x, F(y(t)))$ and $X(t) \succ 0$.

For $x \in (0, \lambda_{\min}(F))$, let $g(x) := \frac{\varepsilon_s x}{n} \text{Tr}(F - xI)^{-1}$. The following claim shows that our choice of δ_s guarantees that $g(\theta)$ is a good approximation of $g(\theta^*) = 1$.

▷ **Claim 7.** $g(\theta(t)) \in (1 - \varepsilon_s, 1)$.

The following two claims show that $\theta(t) \approx X(t) \bullet F(y(t))$ provides a good approximation for $\lambda_{\min}(F(y(t)))$.

▷ **Claim 8.** $(1 - \varepsilon_s)\lambda_{\min}(F(y(t))) < \theta(t) < \frac{\lambda_{\min}(F(y(t)))}{1 + \varepsilon_s/n}$ and $\frac{\lambda_{\min}(F(y(t)))}{1 + \varepsilon_s} \leq \theta^*(t) \leq \frac{\lambda_{\min}(F(y(t)))}{1 + \varepsilon_s/n}$.

▷ **Claim 9.** $\theta(t) < X(t) \bullet F(y(t)) < (1 + \varepsilon_s)\theta(t)$.

Throughout the algorithm, we maintain the invariants that the (non-scaled) dual objective $\mathbf{1}^T y(t)$ is exactly 1, that the step size $\tau(t)$ and the approximation error $\nu(t)$ are between 0 and 1, and that the dual matrix $F(y(t)) = \sum_i y_i(t)A_i$ is positive definite. This is summarized in the following claims.

▷ **Claim 10.** $\mathbf{1}^T y(t) = 1$.

▷ **Claim 11.** For all iterations t , except possibly the last, $\nu(t+1), \tau(t+1) \in (0, 1)$.

▷ **Claim 12.** $F(y(t)) \succ 0$.

4.2.3 Primal Dual Feasibility and Approximate Optimality

Let $t_f + 1$ be the value of t when the algorithm terminates and $s_f + 1$ be the value of s at termination. For simplicity, we write $s = s_f$.

▷ **Claim 13.** (Primal feasibility). $\hat{X} \succ 0$ and $\max_i A_i \bullet \hat{X} \leq 1$.

Proof. The first claim is immediate from Claim 6. To see the second claim, we use the definition of $\nu(t_f)$ and the termination condition in line 4 (which is also satisfied even if $X(t_f) \bullet A_{i(t_f)} - X(t_f) \bullet F(y(t_f)) = 0$):

$$\begin{aligned} \frac{X(t_f) \bullet A_{i(t_f)} - X(t_f) \bullet F(y(t_f))}{X(t_1) \bullet A_{i(t_f)} + X(t_f) \bullet F(y(t_f))} &\leq \varepsilon_s. \\ \therefore (1 + \varepsilon_s)X(t_f) \bullet F(y(t_f)) &\geq (1 - \varepsilon_s)X(t_f) \bullet A_{i(t_f)} \\ &= (1 - \varepsilon_s) \max_i X(t_f) \bullet A_i \\ &\hspace{15em} \text{(by the definition of } i(t_f)) \\ \therefore (1 + \varepsilon_s)^2 \theta(t_f) &\geq (1 - \varepsilon_s) \max_i X(t_f) \bullet A_i. \\ &\hspace{10em} (\because X(t_f) \bullet F(y(t_f)) \leq (1 + \varepsilon_s)\theta(t_f) \text{ by Claim 9}) \end{aligned}$$

The claim follows by the definition of \hat{X} in step 16 of the algorithm. ◁

▷ **Claim 14.** (Dual feasibility). $\hat{y} \geq 0$ and $F(\hat{y}) \succ I$.

Proof. The fact that $\hat{y} \geq 0$ follows from the initialization of $y(0)$ in step 1, Claim 11, and the update of $y(t + 1)$ in step 10. For the other claim, we have

$$\lambda_{\min}(F(\hat{y})) = \frac{1}{\theta(t_f)} \lambda_{\min}(F(y(t_f))) \geq 1 + \frac{\varepsilon_s}{n}. \quad \text{(by Claim 8)}$$

◁

▷ **Claim 15.** (Approximate optimality). $I \bullet \hat{X} \geq \left(\frac{1 - \varepsilon_s}{1 + \varepsilon_s}\right)^2 \mathbf{1}^T \hat{y}$.

Proof. By Claim 7, we have $\text{Tr}(X(t_f)) \geq 1 - \varepsilon_s$, and by Claim 10, we have $\mathbf{1}^T y(t_f) = 1$. The claim follows by the definition of \hat{X} and \hat{y} in step 16. ◁

4.2.4 Running Time per Iteration

Given $F := F(y(t)) \succ 0$, we first compute an approximation $\tilde{\lambda}$ of $\lambda_{\min}(F)$ using Lanczos' algorithm with a random start [25].

► **Lemma 16** ([25]). *Let $M \in \mathbb{S}_+^n$ be a positive semidefinite matrix with N non-zeros and $\gamma \in (0, 1)$ be a given constant. Then there is a randomized algorithm that computes, with high (i.e., $1 - o(1)$) probability a unit vector $v \in \mathbb{R}^n$ such that $v^T M v \geq (1 - \gamma)\lambda_{\max}(M)$. The algorithm takes $O\left(\frac{\log n}{\sqrt{\gamma}}\right)$ iterations, each requiring $O(N)$ arithmetic operations.*

By Claim 8, we need $\tilde{\lambda}$ to lie in the range $\left[\frac{\lambda_{\min}(F)}{1 + \varepsilon_s/n}, \lambda_{\min}(F)\right]$. To obtain $\tilde{\lambda}$, we may apply the above lemma with $M := F^{-1}$ and $\gamma := \frac{\varepsilon_s}{2n}$. Then in $O\left(\sqrt{\frac{n}{\varepsilon_s}} \log n\right)$ iterations we get $\tilde{\lambda} := \frac{1 - \gamma}{v^T F^{-1} v}$ satisfying our requirement. However, we can save (roughly) a factor of \sqrt{n} in

the running time by using, instead, $M := F^{-n}$ and $\gamma := \frac{\varepsilon_s}{2}$. Let v be the vector obtained from Lemma 16, and set $\tilde{\lambda} := \left(\frac{1-\gamma}{v^T F^{-n} v}\right)^{1/n}$. Then, as $\lambda_{\max}(M) \geq v^T M v \geq (1-\gamma)\lambda_{\max}(M)$, and $\lambda_{\min}(F) = \lambda_{\max}(F^{-n})^{-1/n}$, we get

$$\frac{\lambda_{\min}(F)}{1 + \varepsilon_s/n} \leq (1-\gamma)^{1/n} \lambda_{\min}(F) \leq \tilde{\lambda} \leq \lambda_{\min}(F). \quad (13)$$

Note that we can compute F^{-n} in $O(n^\omega \log n)$, where w is the *exponent of matrix multiplication*. Thus, the overall running time for computing $\tilde{\lambda}$ is $O(n^\omega \log n + \frac{n^2 \log n}{\sqrt{\varepsilon_s}})$.

Given $\tilde{\lambda}$, we know by Claim 8 and (13) that $\theta^*(t) \in [\frac{\tilde{\lambda}}{1+\varepsilon_s}, \tilde{\lambda}]$. Then we can apply binary search to find $\theta(t) := \theta^*(t)_{\delta_s}$ as follows. Let $\theta_k = \frac{\tilde{\lambda}}{1+\varepsilon_s}(1+\delta_s)^k$, for $k = 0, 1, \dots, K := \lceil \frac{2 \ln(1+\varepsilon_s)}{\delta_s} \rceil$, and note that $\theta_L \geq \tilde{\lambda}$. Then we do binary search on the exponent $k \in \{0, 1, \dots, K\}$; each step of the search evaluates $g(\theta_k) := \frac{\varepsilon_s \theta_k}{n} \text{Tr}(F - \theta_k I)^{-1}$, and depending on whether this value is less than or at least 1, the value of k is increased or decreased, respectively. The search stops when the search interval $[\ell, u]$ has $u \leq \ell + 1$, in which case we set $\theta(t) = \theta_\ell$; the number of steps until this happens is $O(\log K) = O(\log \frac{1}{\delta_s}) = O(\log \frac{1}{\varepsilon_s})$. By the monotonicity of $g(x)$ (in the interval $[0, \lambda_{\min}(F)]$), and the property of binary search, we know that $\theta^* \in [\theta_\ell, \theta_u]$. Thus, by the stopping criterion,

$$\theta(t) = \theta_\ell \leq \theta^*(t) \leq \theta_u \leq \theta_{\ell+1} = (1 + \delta_s)\theta_\ell,$$

implying that $(1 - \delta_s)\theta^*(t) \leq \theta(t) \leq \theta^*(t)$. Since evaluating $g(\theta_\ell)$ takes $O(n^\omega)$, the overall running time for the binary search procedure is $O(n^\omega \log \frac{n}{\varepsilon_s})$, and hence the total time needed for computing $\theta(t)$ is $O(n^\omega \log \frac{n}{\varepsilon} + \frac{n^2 \log n}{\sqrt{\varepsilon}})$.

All other steps of the algorithm inside the inner while-loop can be done in $O(\mathcal{T} + n^2)$ time, where \mathcal{T} is the time taken by a single call to the oracle $\text{Max}(X(t))$ in step 7 of the algorithm. Thus, in view of Claim 26 on the number of iterations below, we obtain Theorem 1.

4.2.5 Number of Iterations

Define $B = B(t) := \frac{n}{\varepsilon_s \theta} \left(\tau X^{1/2} (\hat{F} - F) X^{1/2} - (\theta^* - \theta) X \right)$. The following (technical) claims are needed for the proofs of Claims 23 and 24 below (which are, in turn, the main claims needed for the analysis of the potential function). They can be skipped at a first reading and recalled when needed.

▷ Claim 17. $(F - \theta^* I)^{-1} = \left(\frac{\varepsilon_s \theta}{n} I - (\theta^* - \theta) X \right)^{-1} X$.

▷ Claim 18. $F' - \theta^* I = (F - \theta I)^{1/2} (I + B) (F - \theta I)^{1/2}$.

▷ Claim 19. $\max_j |\lambda_j(B)| \leq \frac{1}{2}$.

▷ Claim 20. $\theta^*(t) < \lambda_{\min}(F(y(t+1)))$.

▷ Claim 21. if $\nu > \varepsilon_s$, then $\text{Tr}(B) \geq \frac{\nu^2}{8}$.

▷ Claim 22. If $\nu > \varepsilon_s$, then $\text{Tr}(B^2) < \frac{\nu^2}{10}$.

Define the potential function $\Phi(t) := \Phi(\theta^*(t), F(y(t)))$.

The following claim states that the potential difference between two consecutive iterations of the algorithm is sufficiently large.

43:12 Oracle-Based Algorithms for Packing and Covering SDPs

▷ Claim 23. For $t, t+1$ in phase s , $\Phi(t+1) - \Phi(t) \geq \frac{\varepsilon_s \nu(t+1)^2}{40n}$.

Proof. Note that Claim 20 implies that θ^* is feasible to the problem $\max\{\Phi(\xi, F') : 0 \leq \xi \leq \lambda_{\min}(F')\}$. Thus,

$$\begin{aligned}
\Phi(t+1) &= \Phi(\theta^*(t+1), F') \geq \ln \theta^* + \frac{\varepsilon_s}{n} \ln \det(F' - \theta^* I). \\
\therefore \Phi(t+1) - \Phi(t) &\geq \frac{\varepsilon_s}{n} (\ln \det(F' - \theta^* I) - \ln \det(F - \theta^* I)) \\
&\geq \frac{\varepsilon_s}{n} (\ln \det(F' - \theta^* I) - \ln \det(F - \theta I)) && (\because \theta \leq \theta^*) \\
&= \frac{\varepsilon_s}{n} \ln \det(I + B) && (\text{by Claim 18}) \\
&= \frac{\varepsilon_s}{n} \sum_{j=1}^n \ln(1 + \lambda_j(B)) \\
&\geq \frac{\varepsilon_s}{n} \sum_{j=1}^n (\lambda_j(B) - \lambda_j(B)^2) \\
&&& (\text{by Claim 19 and } \ln(1+z) \geq z - z^2, \forall z \geq -0.5) \\
&= \frac{\varepsilon_s}{n} (\text{Tr}(B) - \text{Tr}(B^2)) \\
&> \frac{\varepsilon_s}{8n} \nu^2 - \frac{\varepsilon_s}{10n} \nu^2 && (\text{by Claims 21 and 22}) \\
&= \frac{\varepsilon_s}{40n} \nu^2.
\end{aligned}$$

◁

On the other hand, the following claim states that the overall potential difference between any iterations cannot be too large.

▷ Claim 24. For any t, t' in phase s ,

$$\Phi(t') - \Phi(t) \leq (1 + \varepsilon_s) \ln \frac{X(t) \bullet A_{i(t)}}{(1 - \varepsilon_s) X(t) \bullet F(y(t))}.$$

Proof. Write $F = F(y(t))$, $\theta^* := \theta^*(t)$, $\theta := \theta(t)$, $X := X(t)$, $F' = F(y(t'))$, $\theta'^* := \theta^*(t')$. Then

$$\begin{aligned}
\Phi(t') - \Phi(t) &= \ln \frac{\theta'^*}{\theta^*} + \frac{\varepsilon_s}{n} \ln \det [(F - \theta^* I)^{-1} (F' - \theta'^* I)] \\
&= \ln \frac{\theta'^*}{\theta^*} + \frac{\varepsilon_s}{n} \ln \det \left[\left(\frac{\varepsilon_s \theta}{n} I - (\theta^* - \theta) X \right)^{-1} X (F' - \theta'^* I) \right] && (\text{by Claim 17}) \\
&= \ln \frac{\theta'^*}{\theta^*} + \frac{\varepsilon_s}{n} \left[\ln \det \left(\frac{\varepsilon_s \theta}{n} I - (\theta^* - \theta) X \right)^{-1} + \ln \det [X (F' - \theta'^* I)] \right] \\
&\leq \ln \frac{\theta'^*}{\theta^*} + \frac{\varepsilon_s}{n} \left[\ln \left(\frac{\varepsilon_s \theta}{n} - \frac{\delta_s \theta}{1 - \delta_s} \right)^{-n} + \ln \det [X (F' - \theta'^* I)] \right] \\
&&& (\because \text{Tr}(X) \leq 1 \text{ by Claim 7 and } (1 - \delta_s)\theta^* \leq \theta)
\end{aligned}$$

$$\begin{aligned}
&\leq \ln \frac{\theta'^*}{\theta^*} + \frac{\varepsilon_s}{n} \left[\ln \left(\frac{n}{(1-\varepsilon_s)\varepsilon_s\theta} \right)^n + \ln \det X(F' - \theta'^* I) \right] \\
&\hspace{15em} \text{(by definition of } \delta_s \text{)} \\
&= \ln \frac{\theta'^*}{\theta^*} + \varepsilon_s \ln \frac{n}{(1-\varepsilon_s)\varepsilon_s\theta} + \frac{\varepsilon_s}{n} \ln [\det X(F' - \theta'^* I)] \\
&= \ln \frac{\theta'^*}{\theta^*} + \varepsilon_s \ln \frac{n}{(1-\varepsilon_s)\varepsilon_s\theta} + \frac{\varepsilon_s}{n} \sum_{j=1}^n \ln \lambda_j(X(F' - \theta'^* I)) \\
&\leq \ln \frac{\theta'^*}{\theta^*} + \varepsilon_s \ln \frac{n}{(1-\varepsilon_s)\varepsilon_s\theta} + \varepsilon_s \ln \left(\frac{1}{n} \sum_{j=1}^n \lambda_j(X(F' - \theta'^* I)) \right) \\
&\hspace{15em} \text{(by the concavity of } \ln(\cdot) \text{)} \\
&= \ln \frac{\theta'^*}{\theta^*} + \varepsilon_s \ln \frac{n}{(1-\varepsilon_s)\varepsilon_s\theta} + \varepsilon_s \ln \left(\frac{1}{n} \text{Tr}(X(F' - \theta'^* I)) \right) \\
&\leq \ln \frac{\theta'^*}{\theta^*} + \varepsilon_s \ln \frac{n}{(1-\varepsilon_s)\varepsilon_s\theta} + \varepsilon_s \ln \left(\frac{X \bullet F' - \theta'^*(1-\varepsilon_s)}{n} \right) \\
&\hspace{15em} (\because \text{Tr}(X) \geq 1 - \varepsilon_s \text{ by Claim 7)} \\
&= \ln \frac{\theta'^*}{\theta^*} + \varepsilon_s \ln \frac{1}{(1-\varepsilon_s)\varepsilon_s\theta} + \varepsilon_s \ln (X \bullet F' - \theta'^*(1-\varepsilon_s)) \\
&\leq \ln \frac{\theta'^*}{\theta^*} + \varepsilon_s \ln \frac{1}{(1-\varepsilon_s)\varepsilon_s\theta} + \varepsilon_s \ln \left(\max_{y \in \mathbb{R}_+^n: \mathbf{1}^T y = 1} X \bullet F(y) - \theta'^*(1-\varepsilon_s) \right) \\
&\hspace{15em} (\because \mathbf{1}^T y(t') = 1 \text{ by Claim 10)} \\
&= \ln \frac{\theta'^*}{\theta^*} + \varepsilon_s \ln \frac{1}{(1-\varepsilon_s)\varepsilon_s\theta} + \varepsilon_s \ln (X \bullet A_{i(t)} - \theta'^*(1-\varepsilon_s)) \\
&\hspace{15em} \text{(by definition of } i(t) \text{)} \\
&\leq \max_{0 \leq \xi < X \bullet A_{i(t)}} \left\{ \ln \frac{\xi}{(1-\varepsilon_s)\theta^*} + \varepsilon_s \ln \frac{1}{(1-\varepsilon_s)\varepsilon_s\theta} + \varepsilon_s \ln (X \bullet A_{i(t)} - \xi) \right\} \\
&= (1 + \varepsilon_s) \ln \frac{X \bullet A_{i(t)}}{(1-\varepsilon_s^2)\theta} + \ln \frac{\theta}{\theta^*} \hspace{5em} (\max(\cdot) \text{ is achieved at } \xi = \frac{X \bullet A_{i(t)}}{1+\varepsilon_s}) \\
&\leq (1 + \varepsilon_s) \ln \frac{X \bullet A_{i(t)}}{(1-\varepsilon_s^2)\theta} \hspace{10em} (\because \theta \leq \theta^*) \\
&\leq (1 + \varepsilon_s) \ln \frac{X \bullet A_{i(t)}}{(1-\varepsilon_s)X \bullet F}. \hspace{10em} \text{(by Claim 9)}
\end{aligned}$$

◁

Recall by assumption (B-I) that $\bar{A} := \sum_{i=1}^r A_i \succ 0$.

▷ **Claim 25.** $\frac{X(0) \bullet A_{i(0)}}{X(0) \bullet F(y(0))} \leq \psi := \frac{r \cdot \lambda_{\max}(A_{i(0)})}{\lambda_{\min}(\bar{A})} \leq \frac{r \cdot \max_i \lambda_{\max}(A_i)}{\lambda_{\min}(\bar{A})} \leq n\tau 2\mathcal{L}$.

Proof. Let $X(0) = \sum_{j=1}^n \lambda_j u_j u_j^T$ be the spectral decomposition of $X(0)$. Then,

$$\begin{aligned}
X(0) \bullet A_{i(0)} &= \sum_{j=1}^n \lambda_j A_{i(0)} \bullet u_j u_j^T \leq \sum_{j=1}^n \lambda_j \lambda_{\max}(A_{i(0)}) = \lambda_{\max}(A_{i(0)}) \cdot \text{Tr}(X(0)) \\
X(0) \bullet F(y(0)) &= \sum_{j=1}^n \lambda_j F(y(0)) \bullet u_j u_j^T \geq \frac{1}{r} \sum_{j=1}^n \lambda_j \lambda_{\min}(\bar{A}) = \frac{1}{r} \lambda_{\min}(\bar{A}) \cdot \text{Tr}(X(0)).
\end{aligned}$$

The claim follows. ◁

Now we combine claims 23, 24 and 25 to obtain a bound on the number of iterations.

▷ Claim 26. The algorithm terminates in at most $O(n \log \psi + \frac{n}{\epsilon^2})$ iterations.

► Remark 27. If we do not insist on a sparse dual solution, then we can use the initialization $y(0) \leftarrow \frac{1}{m} \mathbf{1}$ in step 1 in Algorithm 1, and replace ψ in Claim 25, and hence in the running time in Claim 26, by m .

References

- 1 F. Alizadeh. Interior Point Methods in Semidefinite Programming with Applications to Combinatorial Optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995.
- 2 Z. Allen-Zhu, Y. Tat Lee, and L. Orecchia. Using Optimization to Obtain a Width-independent, Parallel, Simpler, and Faster Positive SDP Solver. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 1824–1831, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics.
- 3 Z. Allen-Zhu, Z. Liao, and L. Orecchia. Spectral Sparsification and Regret Minimization Beyond Matrix Multiplicative Updates. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 237–245, New York, NY, USA, 2015. ACM.
- 4 S. Arora, E. Hazan, and S. Kale. Fast Algorithms for Approximate Semidefinite Programming using the Multiplicative Weights Update Method. In *Proc. 46th Symp. Foundations of Computer Science (FOCS)*, pages 339–348, 2005.
- 5 S. Arora, E. Hazan, and S. Kale. The Multiplicative Weights Update Method: a Meta-Algorithm and Applications. *Theory of Computing*, 8(1):121–164, 2012.
- 6 S. Arora and S. Kale. A combinatorial, primal-dual approach to semidefinite programs. In *Proc. 39th Symp. Theory of Computing (STOC)*, pages 227–236, 2007.
- 7 S. Arora and S. Kale. A Combinatorial, Primal-Dual Approach to Semidefinite Programs. *J. ACM*, 63(2):12:1–12:35, May 2016.
- 8 J. Batson, D. Spielman, and N. Srivastava. Twice-Ramanujan Sparsifiers. *SIAM Review*, 56(2):315–334, 2014.
- 9 A. Ben-Tal, L. El Ghaoui, and A.S. Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press, October 2009.
- 10 A. Ben-Tal and A. Nemirovski. Robust optimization - methodology and applications. *Math. Program.*, 92(3):453–480, 2002.
- 11 R. D. Carr and S. Vempala. Randomized metarounding. *Random Struct. Algorithms*, 20(3):343–352, 2002.
- 12 K. Elbassioni and K. Makino. Finding Sparse Solutions for Packing and Covering Semidefinite Programs. *CoRR*, abs/1809.09698, 2018. [arXiv:1809.09698](https://arxiv.org/abs/1809.09698).
- 13 Dan Garber and Elad Hazan. Sublinear Time Algorithms for Approximate Semidefinite Programming. *Math. Program.*, 158(1-2):329–361, July 2016.
- 14 N. Garg and J. Könemann. Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems. *SIAM Journal on Computing*, 37(2):630–652, 2007.
- 15 M. D. Grigoriadis and L. G. Khachiyan. A sublinear-time randomized approximation algorithm for matrix games. *Operations Research Letters*, 18(2):53–58, 1995.
- 16 M. D. Grigoriadis, L. G. Khachiyan, L. Porkolab, and J. Villavicencio. Approximate max-min resource sharing for structured concave optimization. *SIAM Journal of Optimization*, 41:1081–1091, 2001.
- 17 G. Iyengar, D. J. Phillips, and C. Stein. Approximation Algorithms for Semidefinite Packing Problems with Applications to Maxcut and Graph Coloring. In Michael Jünger and Volker Kaibel, editors, *Integer Programming and Combinatorial Optimization (IPCO)*, pages 152–166, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- 18 G. Iyengar, D. J. Phillips, and C. Stein. Feasible and Accurate Algorithms for Covering Semidefinite Programs. In Haim Kaplan, editor, *Algorithm Theory - SWAT 2010*, pages 150–162, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

- 19 G. Iyengar, D. J. Phillips, and C. Stein. Approximating Semidefinite Packing Programs. *SIAM J. on Optimization*, 21(1):231–268, January 2011.
- 20 R. Jain and P. Yao. A Parallel Approximation Algorithm for Positive Semidefinite Programming. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 463–471, 2011.
- 21 R. Jain and P. Yao. A parallel approximation algorithm for mixed packing and covering semidefinite programs. *CoRR*, abs/1201.6090, 2012. [arXiv:1201.6090](https://arxiv.org/abs/1201.6090).
- 22 P. Klein and H.-I. Lu. Efficient Approximation Algorithms for Semidefinite Programs Arising from MAX CUT and COLORING. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing, STOC '96*, pages 338–347, New York, NY, USA, 1996. ACM.
- 23 P. N. Klein and H.-I. Lu. Space-Efficient Approximation Algorithms for MAXCUT and COLORING Semidefinite Programs. In K.-Y. Chwa and O. H. Ibarra, editors, *Algorithms and Computation*, pages 388–398, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.
- 24 Y. T. Lee and H. Sun. An SDP-based Algorithm for Linear-sized Spectral Sparsification. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 678–687, New York, NY, USA, 2017. ACM.
- 25 Z. Leyk and H. Woźniakowski. Estimating a largest eigenvector by Lanczos and polynomial algorithms with a random start. *Numerical Linear Algebra with Applications*, 5(3):147–164, 1999.
- 26 Y. Nesterov. Quality of semidefinite relaxation for nonconvex quadratic optimization. CORE Discussion Papers 1997019, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), March 1997.
- 27 Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, May 2005.
- 28 Y. Nesterov. Smoothing Technique and its Applications in Semidefinite Optimization. *Mathematical Programming*, 110(2):245–259, July 2007.
- 29 Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994.
- 30 R. Peng and K. Tangwongsan. Faster and Simpler Width-independent Parallel Algorithms for Positive Semidefinite Programming. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '12*, pages 101–108, New York, NY, USA, 2012. ACM.
- 31 R. Peng, K. Tangwongsan, and P. Zhang. Faster and Simpler Width-Independent Parallel Algorithms for Positive Semidefinite Programming. *CoRR*, abs/1201.5135, 2016. [arXiv:1201.5135](https://arxiv.org/abs/1201.5135).
- 32 S. A. Plotkin, D. B. Shmoys, and É. Tardos. Fast Approximation Algorithms for Fractional Packing and Covering Problems. In *Proc. 32nd Symp. Foundations of Computer Science (FOCS)*, pages 495–504, 1991.
- 33 M. K. De Carli Silva, N. J. A. Harvey, and C. M. Sato. Sparse Sums of Positive Semidefinite Matrices. *ACM Trans. Algorithms*, 12(1):9:1–9:17, December 2015.
- 34 L. Vandenberghe and S. Boyd. Semidefinite Programming. *SIAM Review*, 38(1):49–95, 1996. URL: <http://www.jstor.org/stable/2132974>.
- 35 N. E. Young. Sequential and Parallel Algorithms for Mixed Packing and Covering. In *Proc. 42nd Symp. Foundations of Computer Science (FOCS)*, pages 538–546, 2001.

Online Disjoint Set Cover Without Prior Knowledge

Yuval Emek

Technion, Haifa, Israel
yemek@technion.ac.il

Adam Goldbraikh

Technion, Haifa, Israel
sgoadam@campus.technion.ac.il

Erez Kantor

Akamai, Cambridge, Massachusetts, USA
erez.kantor@gmail.com

Abstract

The *disjoint set cover (DSC)* problem is a fundamental combinatorial optimization problem concerned with partitioning the (hyper)edges of a hypergraph into (pairwise disjoint) clusters so that the number of clusters that cover all nodes is maximized. In its *online* version, the edges arrive one-by-one and should be assigned to clusters in an irrevocable fashion without knowing the future edges. This paper investigates the *competitiveness* of online DSC algorithms. Specifically, we develop the first (randomized) online DSC algorithm that guarantees a poly-logarithmic ($O(\log^2 n)$) competitive ratio without prior knowledge of the hypergraph's minimum degree. On the negative side, we prove that the competitive ratio of any randomized online DSC algorithm must be at least $\Omega(\frac{\log n}{\log \log n})$ (even if the online algorithm does know the minimum degree in advance), thus establishing the first lower bound on the competitive ratio of randomized online DSC algorithms.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases disjoint set cover, online algorithms, competitive analysis, competitiveness with high probability

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.44

Funding *Yuval Emek*: The work of Y. Emek was supported in part by an Israeli Science Foundation grant number 1016/17.

1 Introduction

1.1 Model and Problem Statement

A *hypergraph* $G = (V, E)$ consists of a set V of *nodes* and a multiset E of *hyperedges* (or simply *edges*), where each edge is a non-empty subset of V .¹ Unless stated otherwise, we denote $n = |V|$ and $m = |E|$.

The input to the *disjoint set cover (DSC)* problem is a hypergraph $G = (V, E)$ and the output is a *color* assignment $C : E \rightarrow \mathbb{Z}_{>0}$ to the edges in E . The objective is to maximize the number of colors $c \in \mathbb{Z}_{>0}$ that *cover* V , where color c is said to cover V (a.k.a. a *covering* color) if the union over all edges $e \in E$ with color $C(e) = c$ equals V . (The DSC problem

¹ The problem we address in this paper is often defined in terms of the equivalent *set system* terminology, where the nodes in V are identified with the elements of some abstract universe and the edges in E are simply referred to as sets or subsets.



should not be confused with the classic hypergraph *edge coloring* problem that also involves assigning colors to the edges. In particular, in the DSC context, it is not required that edges with the same color are disjoint.)

In the *online DSC* problem, the nodes in V are known in advance while the edges in E , assumed hereafter to be totally ordered with edges indexed as $E = \{e_1, \dots, e_m\}$, arrive sequentially in an online fashion so that edge e_t , $1 \leq t \leq m$, arrives at time t . An online DSC algorithm should decide on the color $C(e_t)$ at (or immediately after) time t , without knowing the future edges e_{t+1}, \dots, e_m , and this decision is irrevocable.

Let $\text{Alg}(G)$ be the number of covering colors obtained by (online or offline) DSC algorithm Alg when invoked on hypergraph G . Following the common practice in the realm of online computation (cf. [4]), we measure the quality of online DSC algorithms by means of *competitive analysis*. A deterministic online DSC algorithm Alg is α -*competitive* if for every n , there exists some $\beta = \beta(n) \geq 0$ such that for every n -node hypergraph G , it holds that

$$\text{Alg}(G) \geq \frac{\text{Opt}(G)}{\alpha} - \beta, \quad (1)$$

where Opt is an optimal offline algorithm. A randomized online DSC algorithm Alg is α -*competitive in expectation* if the bound in (1) holds in expectation; if this bound also holds with high probability (abbreviated *whp*), then Alg is said to be α -*competitive whp*.² We emphasize that these probabilistic statements should hold with respect to the coin tosses of Alg , making no assumptions on the input edge sequence. Notice that since DSC is a maximization problem, it follows that if Alg is α -competitive whp, then it is $O(\alpha)$ -competitive in expectation. We refer to α as the online algorithm's *competitive ratio* and say that this competitive ratio is *pure* if the bound in (1) holds with $\beta = 0$ and *impure* otherwise.

By definition, the *minimum degree* $\delta = \min_{v \in V} |\{e \in E : v \in e\}|$ of hypergraph $G = (V, E)$ serves as an obvious upper bound on $\text{Opt}(G)$. Recalling that E may exhibit edge multiplicities, we emphasize that δ (and $\text{Opt}(G)$) may become arbitrarily large with respect to n as the length m of the input edge sequence increases. To a large extent, this fact is what makes the online DSC problem interesting: if δ would have been bounded as a function of n , then one could have included it in the additive term β and trivially obtain an (impure) competitive ratio of 1.

1.2 Background and Related Work

The DSC problem is a fundamental combinatorial optimization problem with many applications in both the offline and online domains. These applications include scheduling the operation of sensors in sensor networks, allocating servers to users in file systems, and assigning users to tasks in crowd-sourcing platforms; refer to [9] for more details. The offline version of the problem is known to be NP-hard and it can be approximated to within an asymptotically tight $O(\log n)$ approximation ratio [10].

The rigorous study of the online DSC problem was initiated by Pananjady et al. [9].³ They first prove that a deterministic online DSC algorithm that does not hold a prior

² Throughout this paper, we say that event A holds whp if $\mathbb{P}(A) \geq 1 - n^{-z}$ for an arbitrarily large constant z .

³ The authors of [9] also define the DSC problem in terms of a hypergraph, however, in that paper, the role of the nodes and edges is reversed so that the nodes in V arrive in an online fashion, each reporting the edges in E to which it belongs. To avoid confusion, we discuss the results of [9] using the current paper's model that follows the common convention in the literature on online and streaming hypergraph algorithms (see, e.g., [12, 7, 8, 6]), where the hypergraph objects that arrive in an online fashion are the edges in E .

knowledge of the minimum degree δ cannot admit a pure competitive ratio better than $\Omega(n)$.⁴ Following that, Pananjady et al. focus their attention on online algorithms that know δ (or an approximation thereof) in advance and develop a deterministic purely $O(\log n)$ -competitive online DSC algorithm. They also establish an $\Omega(\sqrt{\log n})$ lower bound on the (impure) competitive ratio of any such algorithm.

The DSC problem can be viewed as a (maximization) extension of the classic (minimization) *set cover* problem, where (using our hypergraph terminology) the goal is to construct a covering edge subset of minimum size. In its online version, the hypergraph $G = (V, E)$ is known in advance, but only a subset $V' \subseteq V$ of the nodes should be covered. Those are revealed one-by-one in an online fashion and must be covered immediately upon arrival. Using the *online primal-dual* technique (see [5]), Alon et al. [1] developed a (deterministic) online algorithm for this problem with competitive ratio $O(\log n \log m)$. They also proved that this is optimal up to an $O(\log n + \log m)$ factor.

1.3 Our Contribution

Our goal in this paper is to lift the assumption that the minimum degree δ is known in advance, aiming for online DSC algorithms that do not hold any initial knowledge of that hypergraph parameter, referred to hereafter as δ -oblivious online algorithms. As a warm up, we develop a simple deterministic δ -oblivious online algorithm with linear (in n) pure competitive ratio, thus matching the $\Omega(n)$ lower bound of [9]. Nevertheless, we wish to obtain a sublinear competitiveness which means that our online algorithms must be either randomized or admit an impure competitive ratio (or both). We advocate for this compromise: randomization as well as impure competitiveness are omnipresent in the online computation literature and seem like a small price to pay for lifting the often unrealistic assumption that the online algorithm knows the parameter δ in advance, recalling that this parameter would typically increase with the length of the input sequence.

The main technical contribution of the current paper is twofold: On the positive side, we develop a randomized δ -oblivious online DSC algorithm and prove that it is purely $O(\log^2 n)$ -competitive in expectation and impurely $O(\log^2 n)$ -competitive whp. On the negative side, we prove that no randomized online DSC algorithm can have impure competitive ratio better than $\Omega(\log n / \log \log n)$ in expectation or whp. Interestingly, this result holds even for online algorithms that know δ in advance, thus improving upon the $\Omega(\sqrt{\log n})$ lower bound of [9]. A comparison between the results of [9] and those established in the current paper is presented in Table 1.

1.4 Paper's Organization

The remainder of this paper is organized as follows. Following some preliminaries in Section 2, we present our simple deterministic δ -oblivious online DSC algorithm in Section 3, where we also prove that it is $O(n)$ -competitive. Section 4 is then dedicated to our main positive result: a randomized δ -oblivious online DSC algorithm with competitive ratio $O(\log^2 n)$. The $\Omega(\log(n)/\log \log n)$ lower bound on the competitiveness of randomized online DSC algorithms is established in Section 5. Finally, Section 6 is dedicated to some open questions.

⁴ This negative result is obtained on hypergraphs whose δ parameter is proportional to n and the authors of [9] state it as an $\Omega(\delta)$ lower bound. We prefer to view it as an $\Omega(n)$ lower bound since in the current paper, all competitive ratio bounds are expressed as a function of n , and since it does not rule out the existence of a deterministic online DSC algorithm with pure competitive ratio $O(n)$ that works even for instances with $\delta \gg n$ (see Section 1.3).

■ **Table 1** A comparison between the existing state of the art (top cell in each table entry) and the new results established in the current paper (bottom cell in each table entry). Empty cells indicate the lack of known results or lack of improvement over the existing results.

		<i>known</i> δ		<i>unknown</i> δ	
		<i>up. bound</i>	<i>low. bound</i>	<i>up. bound</i>	<i>low. bound</i>
<i>deterministic</i>	<i>pure</i>	$O(\log n)$	$\Omega(\sqrt{\log n})$		$\Omega(n)$
			$\Omega\left(\frac{\log n}{\log \log n}\right)$	$O(n)$	
	<i>impure</i>	$O(\log n)$	$\Omega(\sqrt{\log n})$		$\Omega(\sqrt{\log n})$
			$\Omega\left(\frac{\log n}{\log \log n}\right)$	$O(n)$	$\Omega\left(\frac{\log n}{\log \log n}\right)$
<i>rand. whp</i>	<i>pure</i>	$O(\log n)$			
			$\Omega\left(\frac{\log n}{\log \log n}\right)$	$O(n)$	$\Omega\left(\frac{\log n}{\log \log n}\right)$
	<i>impure</i>	$O(\log n)$			
			$\Omega\left(\frac{\log n}{\log \log n}\right)$	$O(\log^2 n)$	$\Omega\left(\frac{\log n}{\log \log n}\right)$
<i>rand. in expect.</i>	<i>pure</i>	$O(\log n)$			
			$\Omega\left(\frac{\log n}{\log \log n}\right)$	$O(\log^2 n)$	$\Omega\left(\frac{\log n}{\log \log n}\right)$
	<i>impure</i>	$O(\log n)$			
			$\Omega\left(\frac{\log n}{\log \log n}\right)$	$O(\log^2 n)$	$\Omega\left(\frac{\log n}{\log \log n}\right)$

2 Preliminaries

Consider some hypergraph $G = (V, E)$. Given node $v \in V$, let $E(v) = \{e \in E \mid v \in e\}$ be the set of edges that contain v and define the *degree* of v in G to be the size of this set, denoted by $d(v) = |E(v)|$. Let $\delta = \min_{v \in V} d(v)$ denote the minimum degree in G .

For $1 \leq t \leq m$, let $E_t = \{e_1, \dots, e_t\}$ be the set of edges that arrive up to (including) time t . Let $E_t(v) = E_t \cap E(v)$ and let $d_t(v) = |E_t(v)|$ be the degree of node v in the hypergraph (V, E_t) . Define $\eta_t = \min_{v \in e_t} d_t(v)$ to be the minimum degree, at time t , among the nodes included in edge e_t .

Recall that the goal in the DSC problem is to assign some color $C(e) \in \mathbb{Z}_{>0}$ to each edge $e \in E$. Color $c \in \mathbb{Z}_{>0}$ is said to *cover* node $v \in V$ if $C(e) = c$ for some edge $e \in E(v)$. Cast in this terminology, the objective of the DSC problem is to maximize the number of covering colors, that is, the colors that cover every $v \in V$ (see Section 1.1).

Given two integers $x \leq x'$, let $[x, x']$ denote the set of integers y satisfying $x \leq y \leq x'$ and let $[x] = [1, x]$. We generalize this notation to $x \in \mathbb{R}_{>1}$ by defining $[x] = \lceil x \rceil$. (The notation $[x, x']$ is reserved in the current paper only for integral x and x' .) A $\log(\cdot)$ operator with an unspecified base refers to $\log_2(\cdot)$.

Concentration Bounds

Binary random variables X_1, \dots, X_k are said to be *non-positively correlated* if the following two properties hold for any $I \subseteq [k]$:⁵

- (a) $\mathbb{P}(\bigwedge_{i \in I} X_i = 0) \leq \prod_{i \in I} \mathbb{P}(X_i = 0)$; and
- (b) $\mathbb{P}(\bigwedge_{i \in I} X_i = 1) \leq \prod_{i \in I} \mathbb{P}(X_i = 1)$.

The following theorem, referred to as *Chernoff's bounds for non-positively correlated random variables*, was proved in [11] (see also [3]).

⁵ In some literature, the term *negatively correlated* is used instead of non-positively correlated.

► **Theorem 2.1.** Let X_1, \dots, X_k be non-positively correlated binary random variables and let $0 \leq a_1, \dots, a_k \leq 1$. Let $X = \sum_{i \in [k]} a_i X_i$ and let $\mu = \mathbb{E}(X)$. Then,

- $\mathbb{P}(X \leq (1 - \delta)\mu) \leq \exp(-\delta^2\mu/2)$ for any $0 \leq \delta \leq 1$; and
- $\mathbb{P}(X \geq d) \leq 2^{-d}$ for any $d \geq 6\mu$.

Notice that independent random variables are, in particular, non-positively correlated. Indeed, by replacing the requirement that the random variables X_1, \dots, X_k are non-positively correlated by the requirement that they are independent, one obtains (two of) the classic Chernoff bounds.

3 Warmup: a Deterministic Greedy Algorithm

We begin with a simple δ -oblivious deterministic online DSC algorithm, referred to as **Greedy**, whose competitive ratio is purely $O(n)$, thus matching the $\Omega(n)$ lower bound of [9] for such algorithms. For each color $c \in \mathbb{Z}_{>0}$, the algorithm maintains the variable $U_t(c)$ defined to be the set of all nodes covered by the edges in E_t whose color is c , that is, $U_t(c) = \bigcup_{1 \leq t' \leq t: C(e_{t'})=c} e_{t'}$.

Greedy uses the $U_{t-1}(\cdot)$ variables to decide on the color assignment of edge e_t , $1 \leq t \leq m$, setting $C(e_t)$ to be the smallest color $c \in \mathbb{Z}_{>0}$ such that $e_t \not\subseteq U_{t-1}(c)$. This can be viewed as coloring e_t with the smallest color whose cover “benefits” from this assignment. The analysis of **Greedy**’s competitive ratio relies on the following two observations.

► **Observation 3.1.** If $\delta \geq 1$, then $\text{Greedy}(G) \geq 1$.

Proof. Follows immediately from the greedy nature of the algorithm that colors edge e_t with color $C(e_t) = 1$ if e_t contains a node that does not belong to any edge e_1, \dots, e_{t-1} . ◀

► **Observation 3.2.** **Greedy** colors at most n edges with color c for every $c \in \mathbb{Z}_{>0}$.

Proof. If edge e_t is assigned with color c , then $|U_t(c)| \geq |U_{t-1}(c)|$. The assertion follows since $U_t(c) \subseteq V$ for every $1 \leq t \leq m$. ◀

We are now ready to prove the following theorem.

► **Theorem 3.3.** **Greedy** is purely $O(n)$ -competitive.

Proof. If $d(v) = 0$ for some node $v \in V$, then clearly $\text{Greedy}(G) = \text{Opt}(G) = 0$, so assume hereafter that $\delta \geq 1$. We argue that $\text{Greedy}(G) \geq \lfloor \delta/n \rfloor$. Combined with Observation 3.1, this implies that

$$\text{Greedy}(G) \geq \max\{1, \delta/n - 1\} \geq \max\{1, \text{Opt}(G)/n - 1\} \geq \text{Opt}(G)/(2n),$$

thus establishing the assertion. To that end, consider some node $v \in V$ and recall that Observation 3.2 ensures that each color $c \in \mathbb{Z}_{>0}$ is assigned to at most n edges in $E(v)$. Therefore, there must exist at least $\lfloor d(v)/n \rfloor \geq \lfloor \delta/n \rfloor$ colors $c \in \mathbb{Z}_{>0}$ that cover v . Due to the greedy nature of the algorithm, we deduce that the colors $1, \dots, \lfloor \delta/n \rfloor$ cover v which establishes the assertion since this is true for every $v \in V$. ◀

4 The Main Algorithm

In this section, we present our main positive contribution: a randomized δ -oblivious online DSC algorithm, referred to as **Ob1v**. We start by providing an intuitive overview for this algorithm in Section 4.1. The algorithm itself is then presented in Section 4.2. In Section 4.3, we establish a combinatorial lemma regarding the online DSC problem in general. This lemma plays a key role in Section 4.4, where we prove that **Ob1v** is $O(\log^2 n)$ -competitive in expectation. Finally, the proof presented in Section 4.4 is extended in Section 4.5 to show that the same (asymptotic) competitive ratio bound holds also whp.

4.1 Technical Challenges and Intuition

Pananjady et al. [10] developed a randomized *offline* DSC algorithm that on hypergraph $G = (V, E)$, simply colors each edge $e \in E$ by a color $C(e)$ picked uniformly at random (abbreviated hereafter by *uar*) from the *palette* $P = [\Theta(\delta/\log n)]$. Since the degree $d(v)$ of every node $v \in V$ is at least δ , it is easy to see that each color $c \in P$ covers v whp, hence, by the union bound, c covers all V whp. Using standard arguments, one can conclude that the expected number of covering colors is at least $\Omega(|P|) = \Omega(\delta/\log n)$, which is an $O(\log n)$ -approximation as $\delta \geq \text{Opt}(G)$.

In [9], Pananjady et al. observed that the offline algorithm of [10] can be implemented as an online algorithm assuming that δ is known in advance. Their main technical contribution was then to derandomize this randomized algorithm by employing the method of conditional expectation (see, e.g., [2]), carefully adjusted to work in an online fashion.

In contrast, in the current paper we aim for a δ -oblivious online algorithm and hence, cannot use $P = [\Theta(\delta/\log n)]$ as the palette from which a color is picked for each edge $e_t \in E$. Instead, we estimate δ by the parameter $\eta_t = \min_{v \in e_t} d_t(v)$ that can be calculated at time t as it depends only on information that was already exposed to the algorithm. The combinatorial key to our algorithm is that (at least) a constant fraction of the edges e_t that contain node $v \in V$ satisfy $\eta_t \geq \Omega(d(v)/n)$. This means that we can identify (in hindsight) a sufficiently large subset of the edges $e_t \in E(v)$ for which $\Omega(\delta/n) \leq \eta_t \leq \delta$, or equivalently, $\log \eta_t \leq \log \delta \leq \log \eta_t + O(\log n)$.

We rely on this combinatorial insight for the design of **Ob1v**: Upon arrival of edge e_t , the algorithm assigns the variable r_t to be an integer picked uar from the integers in the range $[\log \eta_t, \log \eta_t + O(\log n)]$, thus ensuring that 2^{r_t} is a constant approximation of δ with probability $\Omega(1/\log n)$. The algorithm then uses 2^{r_t} to construct the palette $P_t = [\Omega(2^{r_t}/\log^2 n)]$ from which the color $C(e_t)$ of edge e_t is picked (uar), where the role of the extra $\log n$ factor in the denominator is to account for the probability that 2^{r_t} is a good estimate for δ . The rest of the analysis follows the aforementioned line of arguments, concluding that **Ob1v** is $O(\log^2 n)$ -competitive in expectation.

For whp competitiveness we have to work a little bit harder though. While we identify (in hindsight) a palette P of size $\Theta(\delta/\log^2 n)$ such that each color $c \in P$ covers V whp, the number of such colors may be too large to apply the union bound over all of them, thus we cannot simply argue that all colors in P cover V (simultaneously) whp. Instead, we show that for each node $v \in V$, the random variables that indicate the events that color $c \in P$ does not cover v are non-positively correlated. By applying the Chernoff bound for non-positively correlated random variables, we conclude that at most a $(1/(2n))$ -fraction of the colors in P do not cover v whp, hence the total number of colors in P that do not cover the whole of V is at most $|P|/2$ whp.

4.2 Algorithm's Description

Algorithm `Ob1v` works as follows. Upon arrival of edge e_t , $1 \leq t \leq m$, the algorithm calculates $\eta_t = \min_{v \in e_t} d_t(v)$ and $\ell_t = \lceil \log \eta_t \rceil$ and then assigns the variable r_t to an integer picked uar from the set $[\ell_t, \ell_t + \lceil \log(n-1) \rceil + 2]$. Following that, the color $C(e_t)$ of edge e_t is picked uar from the palette $P_t = \lceil \xi \cdot 2^{r_t} / \log^2 n \rceil$, where $\xi > 0$ is a constant whose value is determined (implicitly) later on. A pseudocode description of `Ob1v` is provided in Algorithm 1.

■ **Algorithm 1** The operation of `Ob1v` upon arrival of edge e_t , $1 \leq t \leq m$.

```

1:  $\eta_t \leftarrow \min_{v \in e_t} d_t(v)$ 
2:  $\ell_t \leftarrow \lceil \log \eta_t \rceil$ 
3: pick  $r_t$  uar from  $[\ell_t, \ell_t + \lceil \log(n-1) \rceil + 2]$ 
4:  $P_t \leftarrow \lceil \xi \cdot 2^{r_t} / \log^2 n \rceil$ 
5: color edge  $e_t$  with color  $C(e_t)$  picked uar from  $P_t$ 

```

▷ $\xi > 0$ is a constant

4.3 A Combinatorial Lemma

Fix some node $v \in V$. Edge $e_t \in E(v)$ is said to be *heavy* (for v) if

$$d_t(v) \leq 2(n-1)\eta_t;$$

otherwise, we say that it is *light* (for v).

► **Lemma 4.1.** *For every time $1 \leq T \leq m$, more than $d_T(v)/2$ of the edges in $E_T(v)$ are heavy.*

Proof. Consider the edge sequence $\sigma = (e_1, \dots, e_T)$. By the definition of η_t , edge $e_t \in E_T(v)$ is light if and only if there exists some node $u \in e_t - \{v\}$ whose degree at time t satisfies $d_t(u) < d_t(v)/(2(n-1))$. On an intuitive level, this means that the challenge in constructing an edge sequence σ that contradicts the assertion, is to increase the degree of v while keeping the degrees of the other nodes small, thus enabling the generation of many light edges with few heavy edges. We employ this intuition to make the following simplifying assumptions.

The first assumption we make for the sake of simplifying the proof is that v is contained in all edges of the sequence σ , that is, $E_T(v) = E_T$. This assumption is clearly without loss of generality since the existence of an edge e_t that does not contain v (and hence is neither heavy nor light) increases the degrees of the nodes in e_t without increasing the degree of v .

Next, notice that all singleton edges of the form $e_t = \{v\}$ are heavy. The second assumption we make for the sake of simplifying the proof is that every heavy edge e_t in σ is a singleton, i.e., $e_t = \{v\}$. To see that this assumption is without loss of generality, suppose that e_t includes additional nodes $u \neq v$ and consider the edge sequence σ' obtained from σ by removing these nodes u from e_t . Comparing σ' to σ , one observes that $d_{t'}(u)$ decreases and $d_{t'}(v)$ remains unchanged for every $t' \geq t$, thus if σ contradicts the assertion, then so does σ' .

The third assumption we make for the sake of simplifying the proof is that every light edge e_t in σ is of size $|e_t| = 2$. To see that this assumption is without loss of generality, suppose that e_t is light with $|e_t| \geq 3$ and let u be a node of minimum degree $d_t(u)$ in e_t . Consider the edge sequence σ' obtained from σ by removing from e_t any node $u' \in e_t - \{v, u\}$. Comparing σ' to σ , one observes that edge e_t remains light (due to the existence of u) while $d_{t'}(u')$ decreases and $d_{t'}(v)$ remains unchanged for every $t' \geq t$, thus if σ contradicts the assertion, then so does σ' .

The fourth assumption we make for the sake of simplifying the proof is that σ is composed of a prefix of heavy edges followed by a suffix of light edges; i.e., there exists some $1 \leq \hat{t} \leq T$ such that edge e_t is heavy if $1 \leq t \leq \hat{t}$ and light if $\hat{t} + 1 \leq t \leq T$. To see that this assumption is without loss of generality, suppose that there exists some $1 \leq t \leq T - 1$ such that edge $e_t = \{v, u\}$ is light and edge $e_{t+1} = \{v\}$ is heavy and consider the edge sequence σ' obtained from σ by swapping between e_t and e_{t+1} . By construction, this swap does not turn e_t (now arriving at time $t + 1$) into a heavy edge as $d_u(t + 1) = d_u(t)$ and $d_v(t + 1) > d_v(t)$, thus if σ contradicts the assertion, then so does σ' . Our assumption is now justified by repeating these swap operations.

So, based on the aforementioned four assumptions, the edge sequence $\sigma = (e_1, \dots, e_T)$ consists of a prefix $(e_1, \dots, e_{\hat{t}})$ of heavy edges of the form $e_t = \{v\}$ and a suffix $(e_{\hat{t}+1}, \dots, e_T)$ of light edges of the form $e_t = \{v, u\}$ for some node $u \neq v$ referred to hereafter as the *extra* node of edge e_t . The fifth and last assumption we make for the sake of simplifying the proof is that the degrees of the extra nodes are monotonically non-decreasing; that is, if u is the extra node of edge e_t , $\hat{t} + 1 \leq t \leq T - 1$, and u' is the extra node of edge e_{t+1} , then $d_t(u) \leq d_{t+1}(u')$. To see that this assumption is without loss of generality, suppose that $d_t(u) > d_{t+1}(u')$ and consider the edge sequence σ' obtained from σ by swapping between e_t and e_{t+1} . By construction, since e_t and e_{t+1} are light in σ , they are also light in σ' , thus if σ contradicts the assertion, then so does σ' . Our assumption is now justified by repeating these swap operations.

Observe that the last simplifying assumption implies that if u is the extra node of edge e_t , $\hat{t} + 1 \leq t \leq T$, and there exists some node $u' \notin \{v, u\}$ with $d_{t-1}(u') < d_{t-1}(u)$, then u' does not appear as the extra node of any edge $e_{t'}$, $t \leq t' \leq T$. This observation allows us to conclude that if u is the extra node of edge e_t , $\hat{t} + 1 \leq t \leq T$, then $d_t(u) \geq (t - \hat{t})/(n - 1)$.

We are now ready to establish the assertion by proving that $\hat{t} > T/2$. To that end, recall that by the definition of a light edge, if u is the extra node of edge e_t , $\hat{t} + 1 \leq t \leq T$, then

$$d_t(u) < \frac{d_t(v)}{2(n-1)} = \frac{t}{2(n-1)}.$$

Put together with the bound $d_t(u) \geq (t - \hat{t})/(n - 1)$, we conclude that $t/2 > t - \hat{t}$ which holds if and only if $\hat{t} > t/2$, thus completing the proof by taking $t = T$. \blacktriangleleft

► **Corollary 4.2.** *For every time $1 \leq T \leq m$, if $d_T(v) \geq z$, then*

$$\left| \left\{ e_t \in E_T(v) \mid \eta_t > \frac{z}{8(n-1)} \right\} \right| > z/4.$$

Proof. Let $e_{t(1)}, \dots, e_{t(z)}$ be the first z edges in the sequence (e_1, \dots, e_T) that contain node v , ordered so that $t(1) < \dots < t(z)$ (we know that these z edges exist as $d_T(v) \geq z$). Lemma 4.1 ensures that more than $z/2$ of the edges $e_{t(j)}$, $1 \leq j \leq z$, are heavy (for v), hence even if all edges in $\{e_{t(j)} \mid 1 \leq j \leq z/4\}$ are heavy, we still have more than $z/4$ heavy edges in $H = \{e_{t(j)} \mid z/4 < j \leq z\}$. Since $d_{t(j)}(v) = j > z/4$ for every edge $e_{t(j)} \in H$, it follows that more than $z/4$ of the edges $e_{t(j)} \in H$ satisfy $\eta_{t(j)} > z/(8(n - 1))$, thus establishing the assertion. \blacktriangleleft

4.4 Competitiveness in Expectation

We now turn to bound the competitive ratio of `Ob1v` in expectation, based on Corollary 4.2. Let $w = \lfloor \log \delta \rfloor$ and let

$$P = \lceil \xi \cdot 2^w / \log^2 n \rceil$$

be the palette from which `Ob1v` picks a color $C(e_t)$ (uar) when the random variable r_t is assigned to $r_t = w$. Given node $v \in V$, let

$$T(v) = \min \{1 \leq t \leq m \mid d_t(v) = 2^w\}$$

be the first time t at which the degree of v reaches $2^w \leq \delta$. Let

$$F(v) = \left\{ e_t \in E_{T(v)}(v) \mid \eta_t > \frac{2^w}{8(n-1)} \right\},$$

recalling that $E_{T(v)}(v)$ is the set of edges $e_t \in E(v)$ with $1 \leq t \leq T(v)$, and let

$$F^w(v) = \{e_t \in F(v) \mid r_t = w\}$$

be the (random) set of edges e_t in $F(v)$ for which `Ob1v` picks a color (uar) from the palette P .

► **Lemma 4.3.** *If $\delta \geq \Omega(\log^2 n)$, then $|F^w(v)| \geq \Omega(\delta/\log n)$ whp for every node $v \in V$.*

Proof. Consider some edge $e_t \in F(v)$ and let A_t denote the event $e_t \in F^w(v)$. By definition,

$$\frac{2^w}{8(n-1)} < \eta_t \leq d_t(v) \leq 2^w,$$

hence

$$w - (\log(n-1) + 3) < \log \eta_t \leq w.$$

Since w is an integer, it follows that

$$w - (\lceil \log(n-1) \rceil + 2) \leq \ell_t \leq w,$$

where recall that $\ell_t = \lceil \log \eta_t \rceil$. Therefore, $w \in [\ell_t, \ell_t + \lceil \log(n-1) \rceil + 2]$ and by the design of the algorithm, we conclude that r_t is assigned to w with probability $1/(\lceil \log(n-1) \rceil + 3)$ implying that $\mathbb{P}(A_t) \geq \Omega(1/\log n)$.

Corollary 4.2 guarantees that $|F(v)| > 2^w/4 \geq \Omega(\delta)$, hence

$$\mathbb{E}(|F^w(v)|) = \sum_{e_t \in F(v)} \mathbb{P}(A_t) \geq \Omega(\delta/\log n).$$

If $\delta \geq \Omega(\log^2 n)$, then $\mathbb{E}(|F^w(v)|) \geq \Omega(\log n)$, therefore, as the events A_t are independent, we can apply Theorem 2.1 to conclude that $|F^w(v)| \geq \Omega(\delta/\log n)$ whp. ◀

► **Corollary 4.4.** *Fix some color $c \in P$. If $\delta \geq \Omega(\log^2 n)$, then c covers v whp for every node $v \in V$.*

Proof. Lemma 4.3 ensures that $|F^w(v)| \geq \Omega(\delta/\log n)$ whp; condition hereafter on this event. The algorithm is designed so that each edge $e_t \in F^w(v)$ is colored $C(e_t) \leftarrow c$ with probability $1/|P| = \Omega(\log^2(n)/\delta)$. Therefore, the probability that none of the edges in $F^w(v)$ is colored c is at most

$$(1 - \Omega(\log^2(n)/\delta))^{\Omega(\delta/\log n)} \leq \exp(-\Omega(\log n)),$$

thus establishing the assertion. ◀

We are now ready to establish the desired competitive ratio bound.

44:10 Online Disjoint Set Cover Without Prior Knowledge

► **Theorem 4.5.** *Ob1v is (impurely) $O(\log^2 n)$ -competitive in expectation.*

Proof. We prove the assertion by showing that

$$\mathbb{E}(\text{Ob1v}(G)) \geq \Omega(\text{Opt}(G)/\log^2 n) - 1.$$

This bound holds trivially if $\text{Opt}(G) \leq \delta \leq O(\log^2 n)$, so assume that $\delta \geq \Omega(\log^2 n)$. Applying the union bound to the promise of Corollary 4.4, we conclude that color $c \in P$ covers all nodes in V whp and, in particular, with probability at least (say) $1/2$. Therefore, by the linearity of expectation,

$$\mathbb{E}(\text{Ob1v}(G)) \geq |P|/2 \geq \Omega(\delta/\log^2 n) \geq \Omega(\text{Opt}(G)/\log^2 n), \quad (2)$$

thus completing the proof. ◀

Recall that in Section 3, we presented the deterministic online DSC algorithm **Greedy** whose competitive ratio is purely $O(n)$. By combining it with **Ob1v**, we can turn the competitive ratio bound promised by Theorem 4.5 into a pure one. To that end, consider the online algorithm Ob1v_p that runs **Ob1v** with probability $1/2$ and **Greedy** with probability $1/2$. If $\delta = 0$, then clearly $\text{Opt}(G) = \text{Ob1v}_p(G) = 0$. If $\delta \geq \Omega(\log^2 n)$, then

$$\mathbb{E}(\text{Ob1v}_p(G)) \geq \mathbb{E}(\text{Ob1v}(G))/2 \geq \Omega(\text{Opt}(G)/\log^2 n),$$

where the last transition holds due to (2). If $1 \leq \delta \leq O(\log^2 n)$, then

$$\mathbb{E}(\text{Ob1v}_p(G)) \geq \text{Greedy}(G)/2 \geq 1/2 \geq \Omega(\text{Opt}(G)/\log^2 n),$$

where the second transition holds due to Observation 3.1. Put together, we obtain the following corollary.

► **Corollary 4.6.** *Ob1v_p is purely $O(\log^2 n)$ -competitive in expectation.*

4.5 Competitiveness with High Probability

We now turn to show that the competitive ratio bound established in Section 4.4 holds also whp (though not purely). For node $v \in V$ and color $c \in P$, define the random variable $X^v(c)$ to be an indicator for the event that color c does *not* cover v , namely, $X^v(c) = 1$ if and only if $C(e_t) \neq c$ for all edges $e_t \in E(v)$. Recall that Corollary 4.4 guarantees that if $\delta \geq \Omega(\log^2 n)$, then

$$\mathbb{E}(X^v(c)) \leq n^{-z} \quad (3)$$

for an arbitrarily large constant z .

The analysis in this section relies on proving that the random variables $X^v(\cdot)$ are non-positively correlated (Lemma 4.8), based on the following observation.

► **Observation 4.7.** *For every node $v \in V$, color subset $Q \subset P$, and color $c' \in P - Q$, we have*

- (a) $\mathbb{P}\left(\bigwedge_{c \in Q} X^v(c) = 0 \mid X^v(c') = 0\right) \leq \mathbb{P}\left(\bigwedge_{c \in Q} X^v(c) = 0\right)$; and
- (b) $\mathbb{P}\left(\bigwedge_{c \in Q} X^v(c) = 1 \mid X^v(c') = 1\right) \leq \mathbb{P}\left(\bigwedge_{c \in Q} X^v(c) = 1\right)$.

Proof. To see that property (a) holds, notice that if $X^v(c') = 0$, then at least one edge in $E(v)$ is colored c' which means that there is one less edge available for the colors in Q , hence $\mathbb{P}\left(\bigwedge_{c \in Q} X^v(c) = 0\right)$ decreases. To see that property (b) holds, notice that if $X^v(c') = 1$, then none of the edges in P is colored c' which means that there is one less color in P to compete with the colors in Q , hence $\mathbb{P}\left(\bigwedge_{c \in Q} X^v(c) = 1\right)$ decreases. ◀

► **Lemma 4.8.** *For every node $v \in V$, the random variables $X^v(c)$, $c \in P$, are non-positively correlated.*

Proof. Fix some $Q \subseteq P$. We prove that $\mathbb{P}\left(\bigwedge_{c \in Q} X^v(c) = 0\right) \leq \prod_{c \in Q} \mathbb{P}(X^v(c) = 0)$; the proof that $\mathbb{P}\left(\bigwedge_{c \in Q} X^v(c) = 1\right) \leq \prod_{c \in Q} \mathbb{P}(X^v(c) = 1)$ is analogous. To that end, we let $Q = \{c_1, \dots, c_k\}$ and prove by induction on k that

$$\mathbb{P}\left(\bigwedge_{i=1}^k X^v(c_i) = 0\right) \leq \prod_{i=1}^k \mathbb{P}(X^v(c_i) = 0).$$

The case $k = 1$ holds trivially, so assume that the inequality holds for $k - 1$ and develop

$$\begin{aligned} \mathbb{P}\left(\bigwedge_{i=1}^k X^v(c_i) = 0\right) &= \mathbb{P}\left(\bigwedge_{i=1}^{k-1} X^v(c_i) = 0 \mid X^v(c_k) = 0\right) \cdot \mathbb{P}(X^v(c_k) = 0) \\ &\leq \mathbb{P}\left(\bigwedge_{i=1}^{k-1} X^v(c_i) = 0\right) \cdot \mathbb{P}(X^v(c_k) = 0) \\ &\leq \prod_{i=1}^{k-1} \mathbb{P}(X^v(c_i) = 0) \cdot \mathbb{P}(X^v(c_k) = 0) = \prod_{i=1}^k \mathbb{P}(X^v(c_i) = 0), \end{aligned}$$

where the second transition follows from Observation 4.7 and the third transition holds due to the inductive hypothesis. ◀

Assume hereafter that $\delta \geq z'n \log^3 n$ for a sufficiently large constant z' which means that $|P| \geq 2zn \log n$ for a constant z that can be made arbitrarily large. Consider some node $v \in V$ and let $X^v = \sum_{c \in P} X^v(c)$. Applying the linearity of expectation to (3), we deduce that $\mathbb{E}(X^v) \leq |P| \cdot n^{-z}$ for an arbitrarily large constant z . Lemma 4.8 allows us to apply Theorem 2.1, thus obtaining the bound

$$\mathbb{P}\left(X^v \geq \frac{|P|}{2n}\right) \leq 2^{-|P|/(2n)} \leq 2^{-2zn \log n/(2n)} = n^{-z} \tag{4}$$

for an arbitrarily large constant z . We are now ready to establish the desired competitive ratio bound.

► **Theorem 4.9.** *0blv is (impurely) $O(\log^2 n)$ -competitive whp.*

Proof. We prove the assertion by showing that

$$0\text{blv}(G) \geq \Omega(0\text{pt}(G)/\log^2 n) - O(n \log n)$$

whp. This bound holds trivially if $0\text{pt}(G) \leq \delta < z'n \log^3 n$ (recall that z' is a constant), so assume that $\delta \geq z'n \log^2 n$ which means that the bound in (4) holds for every node $v \in V$.

44:12 Online Disjoint Set Cover Without Prior Knowledge

Let X be the random variable that takes on the number of colors in P that do *not* cover V . Notice that by the definition of X^v , we know that $X \leq \sum_{v \in V} X^v$. Applying the union bound over all nodes to (4), we conclude that $X^v < |P|/(2n)$ for all nodes $v \in V$ simultaneously whp; condition hereafter on this event. We can now develop

$$X \leq \sum_{v \in V} X^v < |P|/2$$

which means that $\text{obl}_v(G) > |P|/2$. The assertion follows since $|P| \geq \Omega(\delta/\log^2 n)$. ◀

5 Lower Bound

This section is dedicated to our main negative result: there does not exist a randomized online DSC algorithm with (impure) competitive ratio better than $\Omega(\log(n)/\log \log n)$ in expectation (and thus also whp). This lower bound is derived from the following theorem by Yao's min-max principle.

► **Theorem 5.1.** *For every n_0 and δ_0 , there exist $n \geq n_0$, $\delta \geq \delta_0$, and a distribution \mathcal{D} over n -node hypergraphs with minimum degree δ such that (1) $\text{Opt}(G) = \delta$ for every hypergraph G in the support of \mathcal{D} ; and (2) $\mathbb{E}_{G \sim \mathcal{D}}(\text{Alg}(G)) \leq O\left(\delta \frac{\log \log n}{\log n}\right)$ for any deterministic online DSC algorithm Alg .*

Theorem 5.1 is established in two stages. First, in Section 5.1, we construct the promised distribution \mathcal{D} for the special case that $\delta = \Theta(\log(n)/\log \log n)$ (and $\text{Alg}(G) \leq O(1)$). Then, in Section 5.2, we show how this construction is extended for arbitrarily large values of the parameter δ .

5.1 The Basic Construction

Let $q = 2^{2^z}$ for an arbitrarily large integer z and let $r = q/(2 \log q)$ (an integer by the choice of q). Each hypergraph in the support of \mathcal{D} has 2^q nodes, $q + r$ edges, and minimum degree $\delta = r$. We present the construction of a random hypergraph $G = (V, E)$ in \mathcal{D} and then show that $\text{Opt}(G) = r$, whereas

$$\mathbb{E}_G(\text{Alg}(G)) < 3 \tag{5}$$

for any deterministic online DSC algorithm Alg , thus establishing Theorem 5.1 under the restriction that $\delta = \Theta(\log(n)/\log \log n)$.

The nodes in V are identified with the vectors in $\{0, 1\}^q$. The edges in E arrive in the form of a deterministic prefix e_1^p, \dots, e_q^p followed by a random suffix e_1^s, \dots, e_r^s . The prefix is defined by setting

$$e_i^p = \{v \in \{0, 1\}^q \mid v(i) = 1\}$$

for every $i \in [q]$. For the suffix, we pick a partition $\mathcal{S} = \{S_1, \dots, S_r\}$ of $[q]$ into r equally sized clusters (each of size $|S_\ell| = q/r = 2 \log q$) uniformly among the collection of all such partitions. The suffix is then defined by setting

$$e_\ell^s = \{v \in \{0, 1\}^q \mid v(i) = 0 \text{ for all } i \in S_\ell\}$$

for every $\ell \in [r]$. Refer to Figure 1 for an illustration of the suffix edges.

► **Lemma 5.2.** *The hypergraph G satisfies $\text{Opt}(G) = \delta = r$.*

$$e_1^s = \left\{ \begin{array}{cccccccccccccccc} \times & \times & \times & 0 & 0 & 0 & 0 & 0 & \times & \times & \times & 0 & 0 & \times & \times & 0 \end{array} \right\}$$

$$e_2^s = \left\{ \begin{array}{cccccccccccccccc} 0 & 0 & 0 & \times & \times & \times & \times & \times & 0 & 0 & 0 & \times & \times & 0 & 0 & \times \end{array} \right\}$$

Figure 1 The construction of the suffix edges for $q = 16$ assuming that the random partition $\mathcal{S} = \{S_1, S_2\}$ consists of the clusters $S_1 = \{4, 5, 6, 7, 8, 12, 13, 16\}$ and $S_2 = \{1, 2, 3, 9, 10, 11, 14, 15\}$. The \times symbol represents a 'dont-care' vector entry, i.e., it can be a 0 or a 1.

Proof. Since the zero vector $v_0 = (0, \dots, 0)$ is included in all suffix edges $e_\ell^s, \ell \in [r]$ and is not included in any prefix edge $e_i^p, i \in [q]$, it follows that $\delta \leq d(v_0) = r$. The proof is completed by showing that $\text{Opt}(G) \geq r$, recalling that $\text{Opt}(G) \leq \delta$.

Consider the color assignment $C^* : E \rightarrow [r]$ that colors each suffix edge $e_\ell^s, \ell \in [r]$, by color $C^*(e_\ell^s) = \ell$ and each prefix edge $e_i^p, i \in [q]$, by color $C^*(e_i^p) = \ell(i)$ defined to be the unique $\ell \in [r]$ that satisfies $i \in S_\ell$ (this is well defined since $\mathcal{S} = \{S_1, \dots, S_r\}$ is a partition of $[q]$). We argue that under color assignment C^* , color ℓ covers V for every $\ell \in [r]$. Indeed, if vector $v \in \{0, 1\}^q$ is not included in e_ℓ^s , then $v(i) = 1$ for some $i \in S_\ell$, hence v is included in edge e_i^p . This means that $\ell(i) = \ell$, thus $C^*(e_i^p) = \ell$. The assertion follows. \blacktriangleleft

The rest of this section is dedicated to proving that (5) holds. Fix some deterministic DSC algorithm **Alg**. We assume that **Alg** uses only (a subset of) the colors in $[q]$. To see that this assumption is without loss of generality, notice that if color $c \in \mathbb{Z}_{>0}$ is not assigned to any prefix edge $e_i^p, i \in [q]$, then it cannot cover V since the vector $(1, \dots, 1)$ is not included in any suffix edge.

So, let $C : E \rightarrow [q]$ be the color assignment returned by **Alg**. Color $c \in [q]$ is said to be *heavy*, if it is assigned to at least $q/2$ prefix edges, i.e., $|\{i \in [q] \mid C(e_i^p) = c\}| \geq q/2$; otherwise, it is said to be *light*. By definition, there exists at most 2 heavy colors, so $\text{Alg}(G)$ is bounded from above by 2 plus the number of covering light colors. The proof that (5) holds is completed by the following lemma due to the linearity of expectation as clearly, there are at most q light colors in $[q]$.

Lemma 5.3. *If color $c \in [q]$ is light, then c covers V with probability smaller than $1/q$.*

Proof. Consider some light color c and let $I = \{i \in [q] \mid C(e_i^p) = c\}$. Color c is said to be *ℓ -free*, $\ell \in [r]$, if $S_\ell \not\subseteq I$, that is, if there exists some index $j \in S_\ell$ such that the prefix edge e_j^p is not colored c . It is said to be *free* if it is ℓ -free for all $\ell \in [r]$.

We argue that if c is free, then it does not cover V even if all suffix edges are colored c . To that end, let $b_\ell, \ell \in [r]$, be some index in $S_\ell - I$ (this is well defined since c is ℓ -free) and let $B = \{b_\ell \mid \ell \in [r]\}$. Consider the vector v defined by setting $v(i) = 1$ if $i \in B$; and $v(i) = 0$ otherwise. The vector v is not included in any prefix edge $e_i^p, i \in I$, because $B \cap I = \emptyset$, hence $v(i) = 0$ for all $i \in I$. It is also not included in any suffix edge $e_\ell^s, \ell \in [q]$, because $v(b_\ell) = 1$. Therefore, if c is free, then there exists at least one vector in $\{0, 1\}^q$ that it does not cover. Refer to Figure 2 for an illustration. To complete the proof, we show that c is free with probability greater than $1 - 1/q$. Fix some $\ell \in [q]$ and recall that the cluster S_ℓ is a random subset of $[q]$ of size $q/r = 2 \log q$. For the sake of this proof, we think of S_ℓ as being formed by randomly choosing $2 \log q$ indices from $[q]$ without repetitions; denote these indices by $i_1, \dots, i_{2 \log q}$. By definition, color c is not ℓ -free if and only if $i_j \in I$ for all $1 \leq j \leq 2 \log q$.

$$\begin{aligned}
 e_2^p &= \{ \boxed{\times} \boxed{1} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \} \\
 e_4^p &= \{ \boxed{\times} \boxed{\times} \boxed{\times} \boxed{1} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \} \\
 e_7^p &= \{ \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{1} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \} \\
 e_{11}^p &= \{ \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{1} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \} \\
 e_{14}^p &= \{ \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{\times} \boxed{1} \boxed{\times} \boxed{\times} \} \\
 v &= \{ \boxed{0} \boxed{0} \boxed{1} \boxed{0} \boxed{1} \boxed{0} \boxed{0} \boxed{0} \boxed{0} \boxed{0} \boxed{0} \boxed{0} \boxed{0} \boxed{0} \boxed{0} \boxed{0} \}
 \end{aligned}$$

■ **Figure 2** Building on the example depicted in Figure 1, the color c with $I = \{2, 4, 7, 11, 14\}$ is free: it is 1-free because $5 \in S_1 - I$; it is 2-free because $3 \in S_2 - I$. Therefore, the vector v is not covered by any edge in $\{e_2^p, e_4^p, e_7^p, e_{11}^p, e_{14}^p\} \cup \{e_1^s, e_2^s\}$.

We can now develop

$$\mathbb{P} \left(\bigwedge_{j=1}^{2 \log q} i_j \in I \right) = \prod_{j=1}^{2 \log q} \mathbb{P} \left(i_j \in I \mid \bigwedge_{j'=1}^{j-1} i_{j'} \in I \right) = \prod_{j=0}^{2 \log q - 1} \frac{|I| - j}{q - j} \leq (|I|/q)^{2 \log q}.$$

As c is assumed to be light, we have $|I| < q/2$, hence the probability that c is not ℓ -free is smaller than $(1/2)^{2 \log q} = 1/q^2$. By the union bound over all $\ell \in [r]$, we conclude that the probability that c is not ℓ -free for any (at least one) $\ell \in [r]$ is smaller than $r/q^2 < 1/q$, thus establishing the assertion. ◀

5.2 The Multiplied Construction

In this section, we extend the distribution \mathcal{D} presented in Section 5.1 to a distribution \mathcal{D}_k , where k is an arbitrarily large (positive) integer. Each hypergraph in the support of \mathcal{D}_k has 2^q nodes, $k(q+r)$ edges, and minimum degree $\delta_k = kr$. We present the construction of a random hypergraph $G_k = (V_k, E_k)$ in \mathcal{D}_k and then show that $\text{Opt}(G_k) = kr$, whereas

$$\mathbb{E}_{G_k} (\text{Alg}(G_k)) < 3k \tag{6}$$

for any deterministic online DSC algorithm Alg , thus completing the proof of Theorem 5.1.

Like the construction of $G = (V, E)$ presented in Section 5.1, the nodes in V_k are also identified with the vectors in $\{0, 1\}^q$. The basic idea behind the construction of the edge set E_k is to multiply the edges in E , creating k copies for each one of them. A naive attempt to do so would be to simply introduce k independent instantiations of E one after the other with the hope that the arguments used in Section 5.1 can be applied to each instantiation separately. The problem with this approach is that the prefix edges of the $(j+1)$ -st instantiation arrive after the suffix edges of the j -th instantiation, allowing the online algorithm to “color them together” optimally.

To overcome this obstacle, we design the edge sequence so that (all copies of) the prefix edges arrive before (all copies of) the suffix edges. Specifically, the edges in E_k arrive in the form of a deterministic prefix $e_{1,1}^p, \dots, e_{1,k}^p, e_{2,1}^p, \dots, e_{2,k}^p, \dots, e_{q,1}^p, \dots, e_{q,k}^p$ followed by a random suffix $e_{1,1}^s, \dots, e_{1,k}^s, e_{2,1}^s, \dots, e_{2,k}^s, \dots, e_{r,1}^s, \dots, e_{r,k}^s$, where $e_{i,1}^p, \dots, e_{i,k}^p$ and $e_{\ell,1}^s, \dots, e_{\ell,k}^s$ are k identical copies of the edges e_i^p , $i \in [q]$, and e_ℓ^s , $\ell \in [r]$, respectively, as defined in Section 5.1. We emphasize that the same (random) partition $\mathcal{S} = \{S_1, \dots, S_r\}$ is used to determine all copies of the suffix edges and that this partition is revealed to the online algorithm only after (all copies of) all prefix edges have been colored.

Since G_k is obtained from G by edge multiplicity, it follows that $\delta_k = k\delta = kr$, and by Lemma 5.2, we conclude that $\text{Opt}(G_k) = \delta_k = kr$, so it remains to show that (6) holds. To that end, we employ the same proof scheme as in Section 5.1: Fix some deterministic online DSC algorithm **Alg**. Since a color that is not assigned to any prefix edge does not cover the vector $(1, \dots, 1)$, we assume without loss of generality that **Alg** uses only (a subset of) the colors in $[kq]$. As in Section 5.1, we classify the colors in $[kq]$ according to the number of prefix edges they are assigned to, with heavy colors assigned to at least $q/2$ prefix edges and light colors assigned to less than $q/2$ prefix edges. Lemma 5.3 ensures that each light color covers $V = V_k$ with probability smaller than $1/q$, hence, since there are at most kq light colors, we conclude by the linearity of expectation that the expected gain of **Alg** from all light colors is smaller than k . The proof that (6) holds is completed by noticing that there are at most $kq/(q/2) = 2k$ heavy colors.

6 Discussion

Our investigation of the online DSC problem leaves several interesting open questions. The first one concerns the gap between our $O(\log^2 n)$ upper bound and $\Omega(\log(n)/\log \log n)$ lower bound on the competitive ratio of randomized δ -oblivious online DSC algorithms. Since our lower bound holds for online DSC algorithms that know δ in advance as well, one also wonders about the gap it leaves from the $O(\log n)$ upper bound of Pananjady et al. [9] for such algorithms.

The role of randomization in δ -oblivious online DSC algorithms is also not fully understood yet. While the lower bound of [9] states that a deterministic δ -oblivious online DSC algorithm cannot have a pure competitive ratio better than $\Omega(n)$, we still do not know if this is true also for the impure competitiveness of such online algorithms. In particular, it is not clear if the method of conditional expectation applied by Pananjady et al. [9] to derandomize their online algorithm can be applied also to our randomized online algorithm, especially since the derandomization technique of Pananjady et al. relies heavily on the knowledge of δ .

Finally, recall our assumption that the nodes of the hypergraph, and in particular their number n , are known in advance. While the simple deterministic online algorithm presented in Section 3 can be implemented to operate without this assumption, coming up with such an implementation of the randomized online algorithm of Section 4 seems to be a challenging task. More generally, it would be interesting to design online DSC algorithms that are (initially) oblivious to all “global” parameters of the input hypergraph, including both n and δ .

References

- 1 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The Online Set Cover Problem. *SIAM J. Comput.*, 39(2):361–370, 2009.
- 2 Noga Alon and Joel H. Spencer. *The Probabilistic Method*. Wiley Publishing, 4th edition, 2016.
- 3 Anne Auger and Benjamin Doerr. *Theory of randomized search heuristics: Foundations and recent developments*, volume 1, chapter 1: Analyzing Randomized Search Heuristics: Tools from Probability Theory, pages 1–20. World Scientific, 2011.
- 4 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.
- 5 Niv Buchbinder and Joseph Naor. The Design of Competitive Online Algorithms via a Primal-Dual Approach. *Foundations and Trends in Theoretical Computer Science*, 3(2-3):93–263, 2009.

44:16 Online Disjoint Set Cover Without Prior Knowledge

- 6 Yuval Emek and Adi Rosén. Semi-Streaming Set Cover. *ACM Trans. Algorithms*, 13(1):6:1–6:22, 2016.
- 7 Sudipto Guha, Andrew McGregor, and David Tench. Vertex and Hyperedge Connectivity in Dynamic Graph Streams. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '15, pages 241–247, 2015.
- 8 Dmitry Kogan and Robert Krauthgamer. Sketching Cuts in Graphs and Hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS '15, pages 367–376, 2015.
- 9 Ashwin Pananjady, Vivek Kumar Bagaria, and Rahul Vaze. The online disjoint set cover problem and its applications. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 1221–1229. IEEE, 2015.
- 10 Ashwin Pananjady, Vivek Kumar Bagaria, and Rahul Vaze. Optimally Approximating the Coverage Lifetime of Wireless Sensor Networks. *IEEE/ACM Transactions on Networking*, 25(1):98–111, 2017.
- 11 Alessandro Panconesi and Aravind Srinivasan. Randomized Distributed Edge Coloring via an Extension of the Chernoff–Hoeffding Bounds. *SIAM J. Comput.*, 26(2):350–368, 1997.
- 12 Barna Saha and Lise Getoor. On Maximum Coverage in the Streaming Model & Application to Multi-topic Blog-Watch. In *Proceedings of the SIAM International Conference on Data Mining SDM*, pages 697–708, 2009.

Bayesian Generalized Network Design

Yuval Emek

Faculty of Industrial Engineering and Management, Technion, Haifa, Israel
yemek@technion.ac.il

Shay Kutten

Faculty of Industrial Engineering and Management, Technion, Haifa, Israel
kutten@ie.technion.ac.il

Ron Lavi

Faculty of Industrial Engineering and Management, Technion, Haifa, Israel
ronlavi@ie.technion.ac.il

Yangguang Shi

Faculty of Industrial Engineering and Management, Technion, Haifa, Israel
shiyangguang@campus.technion.ac.il

Abstract

We study network coordination problems, as captured by the setting of *generalized network design* (Emek et al., STOC 2018), in the face of uncertainty resulting from partial information that the network users hold regarding the actions of their peers. This uncertainty is formalized using Alon et al.'s *Bayesian ignorance* framework (TCS 2012). While the approach of Alon et al. is purely combinatorial, the current paper takes into account computational considerations: Our main technical contribution is the development of (strongly) polynomial time algorithms for local decision making in the face of Bayesian uncertainty.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Mathematical optimization; Theory of computation → Algorithm design techniques

Keywords and phrases approximation algorithms, Bayesian competitive ratio, Bayesian ignorance, generalized network design, diseconomies of scale, energy consumption, smoothness, best response dynamics

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.45

Related Version A full version of the paper is available at <https://arxiv.org/abs/1907.00484>.

Funding Yuval Emek: The work of Yuval Emek was supported in part by an Israeli Science Foundation grant number 1016/17.

Shay Kutten: The work of Shay Kutten was supported in part by a grant from the ministry of science in the program that is joint with JSPS and in part by the BSF.

Ron Lavi: The work of Ron Lavi was partially supported by the ISF-NSFC joint research program (grant No. 2560/17).

Yangguang Shi: The work of Yangguang Shi was partially supported at the Technion by a fellowship of the Israel Council for Higher Education.

1 Introduction

In real-life situations, network users are often required to coordinate actions for performance optimization. This challenging coordination task becomes even harder in the face of *uncertainty*, as users often act with partial information regarding their peers. Can users overcome their *local* views and reach a good *global* outcome? How far would this outcome be from optimal?



© Yuval Emek, Shay Kutten, Ron Lavi, and Yangguang Shi;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 45; pp. 45:1–45:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

For a formal treatment of the aforementioned questions, we adopt the *Bayesian ignorance* framework of Alon et al. [4]. Consider N agents in a *routing* scenario, where each agent $i \in [N]$ should decide on a (u_i, v_i) -path a_i in the network with the objective of minimizing some global *cost function* that depends on the links' load. The (u_i, v_i) pair, also referred to as the *type* of agent i , is drawn from a distribution p_i . All agents know this distribution, but the actual realization (u_i, v_i) of each agent i is only known to i herself.

Our goal is to construct a *strategy* for each agent i that determines her action a_i based only on her individual type (u_i, v_i) . These strategies are computed in a “preprocessing stage” and the actual decision making happens in real-time without further communication. We measure the quality of a tuple of strategies in terms of its *Bayesian competitive ratio (BCR)* defined as the ratio of the expected cost obtained by these strategies to that of an optimal solution computed by an omnipotent algorithm (refer to Sec. 1.1.1 for the exact definition). To the best of our knowledge, this algorithmic evaluation measure has not been studied so far.

Our main technical contribution is a generic framework that yields strongly polynomial-time algorithms constructing agent strategies with low BCR for *Bayesian generalized network design (BGND)* problems – a setting that includes routing and many other network coordination problems. Our framework assumes cost functions that exhibit *diseconomy of scale (DoS)* [5, 6, 28], capturing the power consumption of devices that employ the popular *speed scaling* technique.

1.1 Model

For clarity of the exposition, we start with the special case of Bayesian routing in Sec. 1.1.1 and then present the more general BGND setting in Sec. 1.1.2. Conceptually, the new algorithmic problem of Bayesian routing that we define here is related to oblivious routing [21, 17, 35], where routing requests should be performed without any knowledge about actual network traffic. This means that the routing path chosen for a routing request may only depend on the network structure and the other parameters of the problem. Oblivious algorithms are attractive as they can be implemented very efficiently in a distributed environment as they base routing decisions only on local knowledge. As will become formally clear below, Bayesian routing has a similar flavor, but with an important additional ingredient. We will assume that the algorithm is equipped with statistical (“Bayesian”) knowledge about network traffic. Thus, in a sense, we replace internal randomization techniques, that oblivious routing usually employs, with actual data, while still being oblivious to other actual routing decisions and thus still maintaining the locality principle.¹

1.1.1 Special Case: Bayesian Routing

In the *full information* variant of the *routing* problem, we are given a (directed or undirected) graph $G = (V, E)$ and a set of N agents, where each agent $i \in [N]$ is associated with a node pair $(u_i, v_i) \in V \times V$, referred to as the (routing) *request* of agent i . This request should be satisfied by choosing some (u_i, v_i) -path in G , referred to as the (feasible) *action* of agent i , and the collection of all such paths is denoted by A_i .

¹ This is different from stochastic network design as these algorithms are not oblivious. More details are given below.

Let $A = A_1 \times \cdots \times A_N$ be the collection of all *action profiles*. The *load* on edge $e \in E$ with respect to action profile $a \in A$, denoted by l_e^a , is defined to be the number of agents whose actions include e , that is, $l_e^a = |\{i \in [N] : e \in a_i\}|$. The cost incurred by load l_e^a on edge e is determined by an (edge specific) *superadditive* cost function $F_e : \mathbb{R}_{\geq 0} \mapsto \mathbb{R}_{\geq 0}$ such that for any $l \geq 0$,

$$F_e(l) = \xi_e \cdot l^\alpha, \quad (1)$$

where $\xi_e > 0$ (a.k.a. the *speed scaling factor*) is a parameter of edge e and $\alpha > 1$ (a.k.a. the *load exponent*) is a global constant parameter. Such a superadditive cost function captures, for example, the power consumption of network devices employing the popular *speed scaling* technique [37, 25, 7, 29, 12, 3] that allows the device to adapt its power level to its actual load. In particular, for those network devices that employ the speed scaling technique, the value of α generally satisfies $1 < \alpha \leq 3$ [24, 36]. Another application of the cost function (1) with $\alpha = 2$ is to model the queuing delay of users in a TCP/IP communication networks [18]. The goal in the (full information) routing problem is to construct an action profile $a \in A$ with the objective of minimizing the total cost $C(a) = \sum_{e \in E} F_e(l_e^a)$.

1.1.1.1 Extending to Partial Information

In the current paper, we extend the full information routing problem to the *Bayesian routing* problem, where the request of agent $i \in [N]$ is not fully known to all other agents. In this problem variant, agent $i \in [N]$ is associated with a set T_i of *types* so that each type $t_i \in T_i$ specifies its own routing request $(u_i^{t_i}, v_i^{t_i}) \in V \times V$. Let $A_i^{t_i}$ be the set of all (feasible) actions for (the request of) type t_i , namely, all $(u_i^{t_i}, v_i^{t_i})$ -paths in G and let $A_i = \bigcup_{t_i \in T_i} A_i^{t_i}$.

Agent i is also associated with a *prior distribution* p_i over the types in T_i and the crux of the Bayesian routing problem is that agent i should decide on her action while knowing the realization of her own prior distribution p_i (that is, the routing request she should satisfy) but without knowing the realizations of the prior distributions of the other agents $j \neq i$. Formally, let $T = T_1 \times \cdots \times T_N$ be the collection of *type profiles* and $A = A_1 \times \cdots \times A_N$ be the collection of *action profiles*. The set of (feasible) action profiles for a type profile $t \in T$ is denoted by $A^t = A_1^{t_1} \times \cdots \times A_N^{t_N}$ and the prior distribution over the type profiles in T is denoted by p . In this paper, p is assumed to be a product distribution, i.e., the probability of type profile $t \in T$ is $p(t) = \prod_{i=1}^N p_i(t_i)$.

The goal in the Bayesian routing problem is to construct for each agent $i \in [N]$, a *strategy* $s_i : T_i \mapsto A_i$ that maps agent i 's realized type $t_i \in T_i$ to an action $a_i \in A_i^{t_i}$. We emphasize that the decision of agent i is taken irrespective of the other agents' realized types which are not (fully) known to agent i . Intuitively, a strategy s_i can be viewed as a *lookup table* constructed in the “preprocessing stage”, and queried at real-time to determine a (fixed) path for every (u_i, v_i) pair associated with i (cf. *oblivious routing* [30, 35]).

The set of strategies available for agent i is denoted by S_i and $S = S_1 \times \cdots \times S_N$ denotes the set of *strategy profiles*. For each type profile $t \in T$, the strategy profile $s \in S$ determines an action profile $a = s(t) \in A$ defined so that $a_i = s_i(t_i)$, $i \in [N]$. Using this notation, the objective in the Bayesian routing problem is to construct a strategy profile $s \in S$ that minimizes the total cost $C(s) = \mathbb{E}_{t \sim p} [\sum_{e \in E} F_e(l_e^{s(t)})]$.

1.1.1.2 Bayesian Competitive Ratio

Consider an algorithm \mathcal{A} that given a Bayesian routing instance, constructs a strategy profile s . To evaluate the performance of \mathcal{A} , we compare the total cost $C(s)$ to $\mathbb{E}_{t \sim p} [\text{OPT}(t)]$, where $\text{OPT}(t) = \min_{a \in A^t} \sum_{e \in E} F_e(l_e^a)$ is the cost of an optimal action profile for the type profile $t \in T$. This can be regarded as the expectation, over the same prior distribution p , of the

total cost incurred by an omnipotent algorithm that has a global view of the whole type profile t and enjoys unlimited computational resources. The *Bayesian competitive ratio* (BCR) of algorithm \mathcal{A} is the smallest $\beta \geq 1$ such that for every Bayesian routing instance, the strategy profile s constructed by \mathcal{A} satisfies $C(s) \leq \beta \cdot \mathbb{E}_{t \sim p}[\text{OPT}(t)]$.

Alon et al. [4] introduced the related criterion of *Bayesian ignorance* defined as $\frac{C(s^*)}{\mathbb{E}_{t \sim p}[\text{OPT}(t)]}$, where $s^* = \text{argmin}_{s \in S} C(s)$ is an optimal strategy profile for the given instance. This criterion quantifies the implication of the agents' partial knowledge regarding the global system configuration, irrespective of the computational complexity of constructing this optimal strategy profile. By definition, for any strategy profile $s \in S$, $C(s) = \mathbb{E}_{t \sim p}[\sum_{e \in E} F_e(l_e^{s(t)})] \geq \mathbb{E}_{t \sim p}[\min_{a \in A^t} \sum_{e \in E} F_e(l_e^a)]$, which implies that the Bayesian ignorance is at least 1. Notice that the BCR is equivalent to the product of the *approximation ratio* $\frac{C(s)}{C(s^*)}$ and the Bayesian ignorance, therefore it evaluates the loss caused by both algorithmic (computational complexity) considerations and the absence of the global information. The first contribution of the current paper is cast in the following theorem.

► **Theorem 1.** *For the Bayesian routing problem, there exists an algorithm whose BCR depends only on the load exponent parameter α . This algorithm is fully combinatorial and runs in strongly polynomial time.*

We emphasize that the BCR of the algorithm promised in Theorem 1 is independent of the number of agents N , the underlying graph G , the speed scaling factors ξ_e , $e \in E$, and the probability distribution p . Therefore, as α is assumed to be a constant, so is the BCR.

1.1.2 Bayesian Generalized Network Design

1.1.2.1 Generalized Network Design

The (full information) routing problem has recently been generalized by Emek et al. [15] to the wider family of *generalized network design (GND)* problems. In its full information form (the form considered in [15]), a GND instance is defined over N agents and a set E of *resources*. Each agent $i \in [N]$ is associated with an abstract (not necessarily routing) request characterized by a set $A_i \subseteq 2^E$ of (feasible) actions out of which, some action $a_i \in A_i$ should be selected. As in the routing case, the action profile $a = (a_1, \dots, a_N)$ induces a load of $l_e^a = |\{i \in [N] : e \in a_i\}|$ on each resource $e \in E$ that subsequently incurs a cost of $F_e(l_e^a)$, where $F_e : \mathbb{R}_{\geq 0} \mapsto \mathbb{R}_{\geq 0}$ is a resource specific cost function. The goal is to construct an action profile $a \in A = A_1 \times \dots \times A_N$ with the objective of minimizing the total cost $C(a) = \sum_{e \in E} F_e(l_e^a)$.

The request of agent $i \in [N]$ is said to be *succinctly represented* [15] if its corresponding action set A_i can be encoded using $\text{poly}(|E|)$ bits. Identifying the resource set E with the edge set of an underlying graph G , the routing requests defined in Sec. 1.1.1 are clearly succinctly represented since each A_i corresponds to the set of (u_i, v_i) -paths in G , hence A_i can be encoded by specifying u_i and v_i (and G). Other examples for succinctly represented requests, where the resource set E is identified with the edge set of an underlying (directed or undirected) graph $G = (V, E)$, include:

- *multi-routing* requests in directed or undirected graphs, where given a collection $D_i \subseteq V \times V$ of *terminal* pairs, the action set A_i consists of all edge subsets $F \subseteq E$ such that the subgraph (V, F) admits a (u, v) -path for every $(u, v) \in D_i$; and
- *set connectivity* (resp., *set strong connectivity*) in undirected (resp., directed) graphs, where given a set $T_i \subseteq V$ of *terminals*, the action set A_i consists of all edge subsets $F \subseteq E$ that induce on G a connected (resp., strongly connected) subgraph that spans T_i .

All requests mentioned (implicitly or explicitly) hereafter are assumed to be succinctly represented.

1.1.2.2 Bayesian GND

In the current paper, we extend the (full information) GND setting to *Bayesian GND* (BGND). This extension is analogous to the extension of full information routing to Bayesian routing as defined in Sec. 1.1.1. In particular, agent $i \in [N]$ is now associated with a set T_i of types, where each type $t_i \in T_i$ corresponds to a request whose action set is denoted by $A_i^{t_i}$, and a prior distribution p_i over the types in T_i . A strategy s_i of agent i is a function that maps the agent's realized type $t_i \in T_i$ to an action $s_i(t_i) \in A_i^{t_i}$.

Similarly to the notation introduced in Sec. 1.1.1, let $T = T_1 \times \cdots \times T_N$ be the set of type profiles. Let $A_i = \bigcup_{t_i \in T_i} A_i^{t_i}$ and let $A = A_1 \times \cdots \times A_N$ be the set of action profiles. Let S_i be the set of strategies available for agent i and let $S = S_1 \times \cdots \times S_N$ be the set of strategy profiles. Given a strategy profile $s \in S$ and a type profile $t \in T$, let $a = s(t) \in A$ be the action profile defined so that $a_i = s_i(t_i)$, $i \in [N]$. The goal in the BGND problem is to construct a strategy profile $s \in S$ with the objective of minimizing the total cost

$$C(s) = \mathbb{E}_{t \sim p} \left[\sum_{e \in E} F_e \left(l_e^{s(t)} \right) \right]. \quad (2)$$

The BCR of Algorithm \mathcal{A} is the smallest $\beta \geq 1$ such that for every BGND instance, the strategy profile $s \in S$ constructed by \mathcal{A} satisfies $C(s) \leq \beta \cdot \mathbb{E}_{t \sim p}[\text{OPT}(t)]$, where $\text{OPT}(t) = \min_{a \in A^t} \sum_{e \in E} F_e(l_e^a)$.

1.1.2.3 Generalized Cost Functions

In addition to the generalization of (full information) routing to GND, [15] also generalizes the cost functions defined in Eq. (1) to cost functions of the form

$$F_e(l) = \sum_{j \in [q]} \xi_{e,j} \cdot l^{\alpha_j}, \quad (3)$$

where q is a positive integer, $\xi_{e,j}$ is a positive real for every $e \in E$ and $j \in [q]$, and α_j is a constant real no smaller than 1 for every $j \in [q]$.² We define $\alpha_{\max} = \max_{j \in [q]} \alpha_j$ and assume hereafter that $\alpha_{\max} > 1$. As discussed in [15], this generalization of Eq. (3) is not only interesting from a theoretical perspective, but also makes the model more applicable to practical network energy saving applications. Indeed, in realistic communication networks, a link often consists of several different devices (e.g., transmitter/receiver, amplifier, adapter), all of which are operating when the link is in use. As their energy consumption can vary in terms of the load exponents and speed scaling factors [36], Eq. (3) may often provide a more accurate abstraction of the actual link's power consumption.

1.1.2.4 Action Oracles

For a BGND problem \mathcal{P} , this paper develops a framework which generates an algorithm with BCR $O(\varrho^{\alpha_{\max}})$ when provided with an *action ϱ -oracle* for \mathcal{P} . An action ϱ -oracle with parameter $\varrho \geq 1$ for BGND problem \mathcal{P} (cf. the *reply ϱ -oracles* of [15]) is a procedure that given agent $i \in [N]$, type $t_i \in T_i$, and a *weight* vector $w \in \mathbb{R}_{>0}^E$, generates an action $a_i \in A_i^{t_i}$ such that $\sum_{e \in a_i} w(e) \leq \varrho \cdot \sum_{e \in a'_i} w(e)$ for any action $a'_i \in A_i^{t_i}$. An *exact action oracle* is an action ϱ -oracle with parameter $\varrho = 1$.

² The cost functions considered in [15] have a fixed additional term, capturing the resource's *startup cost*, that makes them even more general. Due to technical difficulties, in the current paper we were not able to cope with this additional term.

Notice that the optimization problem behind the action oracle is *not* a BGND problem: It deals with a *single* type of a *single* agent and the role of the resource cost functions is now taken by the weight vector. These differences often make it possible to implement the action oracle with known (approximation) algorithms.

For example, the Bayesian routing problem, which requires paths between the given node pairs, admit an exact action oracle implemented using, e.g., Dijkstra's shortest path algorithm [14, 19]. In contrast, the BGND problem with set connectivity requests in undirected graphs (P1), the BGND problem with set strong connectivity requests in directed graphs (P2), the BGND problem with multi-routing requests in undirected graphs (P3), and the BGND problem with multi-routing requests in directed graphs (P4) do not admit exact action oracles unless $P = NP$ as these would imply exact (efficient) algorithms for the *Steiner tree*, *strongly connected Steiner subgraph*, *Steiner forest*, and *directed Steiner forest* problems, respectively. However, employing known approximation algorithms for the latter (Steiner) problems, one concludes that BGND problem (P1) admits an action ϱ -oracle for $\varrho \leq 1.39$ [9]; BGND problem (P2) admits an action ν^ϵ -oracle, where ν is the number of terminals [10]; BGND problem (P3) admits an action 2-oracle [1]; and BGND problem (P4) admits an action $k^{1/2+\epsilon}$ -oracle, where k is the number of terminal pairs [11]. This means, in particular, that BGND problems (P1) and (P3) always admit an action ϱ -oracle with a constant approximation ratio ϱ , whereas BGND problems (P2) and (P4) admit such an oracle when ν and k are fixed [1, 10, 11, 9]. The guarantees of our approximation framework are cast in the following theorem.

► **Theorem 2.** *Consider a BGND problem \mathcal{P} with an action ϱ -oracle $\mathcal{O}_{\mathcal{P}}$. When provided access to $\mathcal{O}_{\mathcal{P}}$, the framework proposed in this paper generates an algorithm $\mathcal{A}_{\mathcal{P}}$ whose BCR depends only on the load exponent parameters $\alpha_1, \dots, \alpha_q$ of Eq. (3). This framework is fully combinatorial and runs in strongly polynomial time, hence if $\mathcal{O}_{\mathcal{P}}$ can be implemented to run in strongly polynomial time, then so can $\mathcal{A}_{\mathcal{P}}$.*

Again, we emphasize that the BCR of the algorithm promised in Theorem 2 is independent of the number of agents N , the number of resources $|E|$, the speed scaling factors $\xi_{e,j}$, $j \in [q]$, $e \in E$, and the probability distribution p . Therefore, as $\alpha_1, \dots, \alpha_q$ are assumed to be constants, so is the BCR. Since the Bayesian routing problem admits an exact action oracle, Theorem 1 follows trivially from Theorem 2. Throughout the remainder of this paper, we focus on the BGND framework promised in Theorem 2.

1.2 Related Works

The technical framework that we use is inspired by [15]. Sec. 3 gives a detailed technical overview including a full comparison.

In the full information case, network design problems with superadditive cost functions as defined in Eq. (1) have been extensively studied with the motivation of improving the energy efficiency of networks [5, 6, 28]. To the best of our knowledge, none of these studies has been extended to the Bayesian case.

In the research works on oblivious routing (e.g., [17, 35, 27, 23]), the absence of global information in routing is modeled in an adversarial (non-Bayesian) manner. In particular, oblivious routing assumes that no knowledge about t_{-i} is available when determining every a_i , and the performance of the algorithm is evaluated by means of its *competitive ratio* $\max_{t \in T} \frac{\sum_{e \in E} F_e(l_e^s(t))}{\text{OPT}(t)}$. For the cost function $F_e(l) = l^\alpha$ with $\alpha > 1$, Englert and Räcke [17]

propose an $O(\log^\alpha |V|)$ -competitive oblivious routing algorithm for the scenario where the traffic requests are allowed to be partitioned into fractional flows. Shi et al. [35] prove that for such a cost function, there exists no oblivious routing algorithm with competitive ratio $O\left(|E|^{\frac{\alpha-1}{\alpha+1}}\right)$ when it is required to choose an integral path for every request.

The Bayesian approach is often used in the game theoretic literature to model the uncertainty a player experiences regarding the actions taken by the other players. Roughgarden [32] studies a *routing game* (among other things) in which the players share (equally) the cost of the edges they use and proposes a theoretical tool called *smoothness* to analyze the *price of anarchy* (PoA) of this game in a Bayesian setting, defined as $\frac{\max_{s \in S^{\text{BNE}}} C(s)}{\mathbb{E}_{t \sim p}[\text{OPT}(t)]}$, where S^{BNE} denotes the set of *Bayes-Nash equilibria*. In particular, he proves that with the cost function $F_e(l) = \xi_{e,1} \cdot l + \xi_{e,2} \cdot l^2$, the PoA is bounded by $\frac{5}{2}$. We employ the smoothness toolbox in our algorithmic construction, as further described in Sec. 5 (see also the overview in Sec. 3, as well as the detail in Sec. 6 of the full version [16]).

Alon et al. [4] investigate the Bayesian routing game with a constant cost function $F_e = \xi_e$ and prove that the Bayesian ignorance $\frac{C(s^*)}{\mathbb{E}_{t \sim p}[\text{OPT}(t)]}$ is bounded by $O(N)$ (resp., $O(\log |E|)$) in directed (resp., undirected) graphs $G = (V, E)$. They also introduce game theoretic variants of the Bayesian ignorance notion and analyze them in that game.

To deal with the inherent uncertainty of the demand in realistic networks, many research works have been conducted on *stochastic network design* [22, 13, 31], formulated as a *two-stage stochastic optimization* problem: in the first stage, each link in the network has a fixed cost and the algorithm needs to make decisions to purchase links knowing the probability distribution over the network demands; in the second stage, the network demands are realized (according to the aforementioned probability distribution) and should be satisfied, which may require purchasing additional links, this time with an inflated cost. The objective is to minimize the total cost of the two stages plus a load dependent term, in expectation.

The BGND setting considered in the current paper is different from two-stage stochastic optimization (particularly, stochastic network design) in several aspects, the most significant one is that in BGND, an agent’s strategy should dictate her “complete action” (e.g., a path for routing requests) for every possible type, obliviously of the realized types of the other agents. In particular, one cannot “update” the agents’ actions and purchase additional resources at a later stage to satisfy the realized demands. Moreover, the current paper evaluates the performance of a BGND algorithm by means of its BCR that takes into consideration computational complexity limitations as well as the lack of global information (see Sec. 1.1) whereas the literature on two-stage stochastic optimization typically evaluates algorithms using standard approximation guarantees that accounts only for computational complexity limitations.

In [20], Garg et al. investigate online combinatorial optimization problems where the requests arriving online are drawn independently and identically from a known distribution. As an example, Garg et al. [20] study the online Steiner tree problem on an undirected graph $G = (V, E)$. In this problem, at each step the algorithm receives a terminal that is drawn independently from a distribution over V , and needs to maintain a subset of edges connecting all the terminals received so far.

Our work differs from [20] in following four aspects. First, in the stochastic online optimization problem studied in [20], when each request i arrives, the previous requests $\{1, \dots, i-1\}$ have been realized, and the realization is known. By contrast, in the BGND problem, every agent i needs to be served without knowing the actual realization of the other agents. Second, the cost function studied in [20] maps each resource e to a fixed toll, which

is subadditive in the number of requests using e , while our cost function is superadditive. Third, in the BGND problem with the set connectivity requests, for each agent i , each type t_i is a set of terminals rather than a single terminal, and each action in $A_i^{t_i}$ is a Steiner tree spanning over the set of terminals corresponding to t_i . Fourth, in the BGND problem, each prior distribution p_i is over the types of agent i , while there is no distribution over the agents.

1.3 Paper Organization

The rest of this paper is organized as follows. Sec. 2 introduces some of the concepts employed in our approximation framework together with some notation and terminology. The main challenges that we had to overcome when developing this framework and some of the techniques used for that purpose are discussed in Sec. 3. Sec. 4 is dedicated to a detailed exposition of our approximation framework. Its performance is then analyzed in Sec. 5 using certain game theoretic properties.

2 Preliminaries

We follow the common convention that for an N -tuple $x = (x_1, \dots, x_N)$ and for $i \in [N]$, the notation x_{-i} denotes the $(N-1)$ -tuple $(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_N)$. Likewise, for a Cartesian product $X = X_1 \times \dots \times X_N$ and for $i \in [N]$, the notation X_{-i} denotes the Cartesian product $X_1 \times \dots \times X_{i-1} \times X_{i+1} \times \dots \times X_N$.

2.1 The BGND Game

Given an instance $\mathcal{I} = \langle N, E, \{T_i, p_i\}_{i \in [N]}, \{\xi_{e,j}\}_{e \in E, j \in [q]}, \{\alpha_j\}_{j \in [q]} \rangle$ of a BGND problem \mathcal{P} , we define a *BGND game* by associating every agent $i \in [N]$ with a strategic player who decides on the strategy s_i with the objective of minimizing her own individual cost defined as follows. Given an action profile $a \in A$ and a resource $e \in E$, the corresponding cost $F_e(l_e^a)$ is equally divided among the players $i \in [N]$ satisfying $e \in a_i$; in other words, the *cost share* of player i in resource e under action profile a , denoted by $f_{i,e}(a)$, is defined to be

$$f_{i,e}(a) = \begin{cases} 0, & e \notin a_i \\ \frac{F_e(l_e^a)}{|\{i: e \in a_i\}|} = \sum_j \xi_{e,j} (l_e^a)^{\alpha_j - 1}, & \text{otherwise} \end{cases}.$$

Informally, the individual cost of player i is the sum of her cost shares over all resources.

For a more formal treatment of the BGND game, we occasionally need to explicitly specify the type t_i of player i in the expressions involving her cost share in which case we use the notation $f_{i,e}(t_i; a)$, following the convention that $f_{i,e}(t_i; a) = f_{i,e}(a)$ if $a_i \in A_i^{t_i}$; and $f_{i,e}(t_i; a) = \infty$ otherwise. The individual cost of a player i with respect to the type t_i and a fixed action profile a is defined as $C_i(t_i; a) = \sum_{e \in E} f_{i,e}(t_i; a)$. Correspondingly, for each player $i \in [N]$ and each type $t_i \in T_i$, we define the *type-specified* expected individual cost $C_i(t_i; s) = \mathbb{E}_{t_{-i} \sim p_{-i}} [C_i(t_i; s(t_i, t_{-i}))]$. The objective function that player i wishes to minimize is her *type-averaged* expected individual cost $C_i(s) = \mathbb{E}_{t_i \sim p_i} [C_i(t_i; s)]$, irrespective of the total cost $C(s)$, often referred to as the *social cost*.

Let $f_{i,e}(a_i; s_{-i}) = \mathbb{E}_{t_{-i} \sim p_{-i}} [f_{i,e}(a_i, s_{-i}(t_{-i}))]$ be the expected cost share of player $i \in [N]$ on resource $e \in E$ with respect to action $a_i \in A_i$ and strategy profile $s_{-i} \in S_{-i}$. Fixing $a_{-i} \in A_{-i}$ (resp., $s_{-i} \in S_{-i}$), the cost share $f_{i,e}(a_i, a_{-i})$ (resp., expected cost share $f_{i,e}(a_i; s_{-i})$) of

player i on resource e is the same for every action $a_i \in A_i$ such that $e \in a_i$. Therefore, it is often convenient to ignore the specifics of action a_i and use the notations $f_{i,e}(+, a_{-i})$ and $\hat{f}_{i,e}(+; s_{-i})$ instead of $f_{i,e}(a_i, a_{-i})$ and $\hat{f}_{i,e}(a_i; s_{-i})$, respectively, given that $e \in a_i$.³

2.2 Definitions for the Algorithm Design and Analysis

The following definitions play key roles in the design and analysis of our framework.

► **Definition (Choice Function [32]).** A choice function $\sigma : T \mapsto A$ maps every type profile $t \in T$ to an action profile $a \in A^t$. The action specified by σ for player $i \in [N]$ with respect to type profile t is denoted by $\sigma_i(t)$. In particular, the choice function that maps each type profile t to an action profile that realizes $\text{OPT}(t)$ is denoted by σ^* .

► **Definition (Smoothness [32]).** Given parameters $\lambda > 0$ and $0 < \mu < 1$, a BGND game is said to be (λ, μ) -smooth if $\sum_{i \in [N]} C_i(t_i; (\sigma_i^*(t), a_{-i})) \leq \lambda \cdot \text{OPT}(t) + \mu \cdot \sum_{i \in [N]} C_i(t'_i, a)$ for every type profiles $t, t' \in T$ and action profile $a \in A^{t'}$.

► **Definition (Potential Function).** A function $\Phi : S \mapsto \mathbb{R}_{\geq 0}$ is said to be a potential function of the BGND game if $\Phi(s) - \Phi(s'_i, s_{-i}) = C_i(s) - C_i(s'_i, s_{-i})$ for every strategy profile $s \in S$, player $i \in [N]$, and strategy $s'_i \in S_i$. The potential function $\Phi(\cdot)$ is said to be K -bounded for a parameter $K \geq 1$ if $\Phi(s) \leq C(s) \leq K \cdot \Phi(s)$ for every strategy profile $s \in S$.

► **Definition ($(\underline{\eta}, \bar{\eta})$ -Estimation).** Given real parameters $\underline{\eta}, \bar{\eta} \geq 1$, a value x is said to be an $(\underline{\eta}, \bar{\eta})$ -estimation of the expected cost share $\hat{f}_{i,e}(a_i; s_{-i})$ (resp., $\hat{f}_{i,e}(+; s_{-i})$) if it satisfies $x/\underline{\eta} \leq \hat{f}_{i,e}(a_i; s_{-i}) \leq x \cdot \bar{\eta}$ (resp., $x/\underline{\eta} \leq \hat{f}_{i,e}(+; s_{-i}) \leq x \cdot \bar{\eta}$). We typically denote this estimation x by $\hat{f}_{i,e}(a_i; s_{-i})$ (resp., $\hat{f}_{i,e}(+; s_{-i})$). The BGND game is said to be poly-time $(\underline{\eta}, \bar{\eta})$ -estimable if for every player $i \in [N]$ and strategy profile $s_{-i} \in S_{-i}$, there exists an algorithm which runs in time $\text{poly}(N, q, |T_1|, \dots, |T_N|)$ and outputs an $(\underline{\eta}, \bar{\eta})$ -estimation of the expected cost share $\hat{f}_{i,e}(+; s_{-i})$. The BGND game is said to be tractable if it is poly-time $(\underline{\eta}, \bar{\eta})$ -estimable with $\bar{\eta} = \underline{\eta} = 1$.

Fix some player $i \in [N]$, type $t_i \in T_i$, and $(\underline{\eta}, \bar{\eta})$ -estimations $\hat{f}_{i,e}(s_i(t_i); s_{-i})$, $e \in E$. With respect to these variables, let $\hat{C}_i(t_i; s) = \sum_{e \in E} \hat{f}_{i,e}(s_i(t_i); s_{-i})$ and $\hat{C}_i(s) = \mathbb{E}_{t_i \sim p_i}[\hat{C}_i(t_i; s)]$. By the linearity of expectation, we know that $\hat{C}_i(t_i; s)/\underline{\eta} \leq C_i(t_i; s) \leq \hat{C}_i(t_i; s) \cdot \bar{\eta}$ and $\hat{C}_i(s)/\underline{\eta} \leq C_i(s) \leq \hat{C}_i(s) \cdot \bar{\eta}$. Consequently, we refer to $\hat{C}_i(t_i; s)$ and $\hat{C}_i(s)$ as $(\underline{\eta}, \bar{\eta})$ -estimations of $C_i(t_i; s)$ and $C_i(s)$, respectively.

► **Definition (Approximate Best Response).** For strategy profile $s \in S$ and player $i \in [N]$, strategy $s_i \in S_i$ is said to be an approximate best response (ABR) of i with approximation parameter $\chi \geq 1$ if $C_i(s_i, s_{-i}) \leq \chi \cdot C_i(s'_i, s_{-i})$ holds for any $s'_i \in S_i$. We may omit the explicit mention of the approximation parameter χ when it is clear from the context. A best response (BR) is an ABR with approximation parameter $\chi = 1$.

► **Definition (Approximate Best Response Dynamics).** An approximate best response dynamic (ABRD) is a procedure that starts from a predetermined strategy profile $s^0 \in S$ and generates a series of strategy profiles s^1, \dots, s^R such that for every $1 \leq r \leq R$, there exists some player $i \in [N]$ satisfying (1) $s_{-i}^r = s_{-i}^{r-1}$; and (2) s_i^r is an ABR of i to s_{-i}^{r-1} .

³ To avoid ambiguity concerning the definition of $f_{i,e}(+, a_{-i})$ and $\hat{f}_{i,e}(+; s_{-i})$ for resources $e \notin A_i$, we assume (in the scope of using these notations) that $A_i = E$ for all $i \in [N]$. This is without loss of generality as one can augment T_i with a virtual type \tilde{t}_i such that $A_i^{\tilde{t}_i} = \{E\}$ and $p_i(\tilde{t}_i)$ is arbitrarily small.

3 Overview of the Main Challenges and Techniques

The approximation framework presented in Sec. 4 for BGND problems is inspired by the framework designed in [15] for full information GND problems only in the conceptual sense that both algorithms employ approximate best response dynamics. In a high-level, for a certain number R of rounds that will be carefully chosen in order to achieve the approximation promise, and starting from some properly chosen initial strategy profile s^0 , for each round $1 \leq r \leq R$ the strategy profile s^r is generated from s^{r-1} in the following manner:

1. For every player $i \in [N]$ and resource $e \in E$, compute an $(\underline{\eta}, \bar{\eta})$ -estimation $\widehat{f}_{i,e}(+; s_{-i}^{r-1})$ of the expected cost share $f_{i,e}(+; s_{-i}^{r-1})$.
2. For every player $i \in [N]$, construct the strategy s'_i by mapping each type $t_i \in T_i$ to the action $a_i \in A_i^{t_i}$ computed by invoking the action ϱ -oracle with weight vector w defined by setting $w(e) = \widehat{f}_{i,e}(+; s_{-i}^{r-1})$.
3. Choose player $i \in [N]$ according to the game theoretic criterion presented in Sec. 4 regarding the estimations $\widehat{C}_i(s^{r-1})$ and $\widehat{C}_i(s'_i, s_{-i}^{r-1})$ of the type-averaged expected individual costs. Construct s^r by updating the strategy of the chosen player i to s'_i .

However, beyond the similar high-level structure, the technical construction in this paper is entirely different from [15] since the incomplete information assumption of the BGND setting exhibits new algorithmic challenges that require novel techniques. Specifically, the main challenges that our technical analysis in this paper handles are as follows.

A first obstacle here is the difficulty in computing the estimation $\widehat{f}_{i,e}(+; s_{-i}^{r-1}) = \mathbb{E}_{t_{-i} \sim p_{-i}}[f_{i,e}(+, s_{-i}(t_{-i}))]$ in step 1 since there are exponentially (in N) many possibilities for t_{-i} . Another source of difficulty in this regard is that the function $f_{i,e}(+, s_{-i}(t_{-i}))$ is nonlinear in $l_e^{s_{-i}(t_{-i})}$. One may hope that Jensen's inequality [26] can resolve this issue, however, as we explain in the technical sections, it is not enough for obtaining proper bounds on both $\underline{\eta}$ and $\bar{\eta}$. This obstacle is addressed in Sec. 5 (and Sec. 8 of the full version [16]) where we employ probabilistic tools from [8] and using Cantelli's inequality [34] to obtain the required estimation of the expression $\mathbb{E}_{t_{-i} \sim p_{-i}}[f_{i,e}(+, s_{-i}(t_{-i}))]$.

A second obstacle is that the ABRD-based approximation framework expresses its approximation guarantees in terms of smoothness parameters and bounded potential functions. However, neither the smoothness parameters nor the existence of a bounded potential function are known for the BGND game that we have defined here. We provide a new analysis for these two issues in Sec. 6 and Sec. 7 of the full version [16], respectively.

A third obstacle involves the stopping condition of the best response dynamics. A stopping condition for the full information case, via the smoothness framework, was developed by [33] (showing that if the current outcome in a best response dynamics is far from optimal there must exist a player whose best response significantly improves his own utility). For the Bayesian case, to the best of our knowledge, no such general stopping condition was known prior to the current paper. In fact, the smoothness framework for the Bayesian case which was developed in [32] did not include any results on best response dynamics. One specific technical difficulty is that Bayesian smoothness is defined in [32] w.r.t. a deviation to the optimal choice function rather than to a best response. This obstacle is resolved in Sec. 5 of the full version [16] where we provide such a stopping condition by proving that if the outcome of the current step of the ABRD in the Bayesian case is far from optimal, there must exist a player whose approximate best response must significantly improve her utility.

A fourth obstacle regards the output of the algorithm, once the ABRD terminates. Although we prove that there exists at least one strategy profile s^r , $1 \leq r \leq R$, with a sufficiently small social cost $C(s^r)$, we do not know how to find it. In particular, we wish to emphasize that we cannot simply evaluate the social cost function $C(\cdot)$ (see Eq. (2)) due

to the exponential number of type profiles. This obstacle does not exist in [15] where they can explicitly go over all steps of the full information ABRD and find the exact step whose outcome has minimal cost. To resolve this issue, we output the last strategy profile s^R generated in the ABRD and bound its loss. This is described in Sec. 5 of the full version [16].

Our technical constructions and our analysis employ various techniques from algorithmic game theory, demonstrating once again (as in [15]) the usefulness of this literature as a toolbox for algorithmic constructions that, on the face of it, have nothing to do with selfish agents. In particular, in this paper (and as assumed in the literature on oblivious routing [21, 17, 35]), we construct an algorithm that receives a correct input and outputs routing tables that the agents are going to follow without issues of selfish deviations.

4 The Algorithm

In this part, we present an algorithm, which is referred to as **Bayes-ABRD**, for a given BGND problem \mathcal{P} . The algorithm is assumed to have free access to an action ϱ -oracle for \mathcal{P} , which is denoted by $\mathcal{O}_{\mathcal{P}}$.

With an input instance $\mathcal{I} = \langle N, E, \{T_i, p_i\}_{i \in [N]}, \{\xi_{e,j}\}_{e \in E, j \in [q]}, \{\alpha_j\}_{j \in [q]} \rangle$, the first step of the algorithm is to (conceptually) construct a BGND game, and choose a tuple of parameters $(\lambda, \mu, K, \underline{\eta}, \bar{\eta})$ such that the BGND game

1. is (λ, μ) -smooth with $\varrho(\underline{\eta}\bar{\eta})^2\mu < 1$,
2. has a potential function Φ that is K -bounded,
3. is poly-time $(\underline{\eta}, \bar{\eta})$ -estimable.

The existence and exact values of the parameters in this tuple are presented in Sec. 5.

► **Lemma 3.** *For any $i \in [N]$ and any $s_{-i} \in S_{-i}$, there exists a $\text{poly}(|E|, N, q, \{|T_i|\}_{i \in [N]})$ -time procedure which generates a strategy $s_i \in S_i$ and the corresponding $(\underline{\eta}, \bar{\eta})$ -estimation $\hat{C}_i(s_i, s_{-i})$ of the individual costs such that $\hat{C}_i(s_i, s_{-i}) \leq \varrho \cdot \eta \cdot C_i(s'_i, s_{-i})$ for any $s'_i \in S_i$. This means in particular that s_i is an ABR of i to s_{-i} with approximation parameter $\varrho \cdot \underline{\eta}\bar{\eta}$.⁴*

Employing the procedure promised by Lemma 3, **Bayes-ABRD** simulates an ABRD of at most R rounds s^0, s^1, \dots for the BGND game induced by \mathcal{I} . Here R is a positive integer depending on the tuple $(\lambda, \mu, K, \underline{\eta}, \bar{\eta})$, and its exact value is also deferred to the following parts (Sec. 5). The ABRD simulated in our algorithm is done as follows.

Each player i chooses her initial strategy s_i^0 by taking each $s_i^0(t_i)$ to be the action generated by $\mathcal{O}_{\mathcal{P}}$ for type t_i with respect to the weight vector w^0 defined by setting $w^0(e) = \sum_{j \in [q]} \xi_{e,j}$, that is, as if i is playing alone. The obtained strategy s_i^0 is broadcast by player i to all the other players such that the full strategy profile s^0 is known by every player. Assuming that s^{r-1} , $1 \leq r \leq R$, was already constructed and known by all the players, s^r is obtained as follows. Every player $i \in [N]$ employs the procedure promised by Lemma 3 to generate an ABR \hat{s}_i^{r-1} to s_{-i}^{r-1} , and computes $\Delta_i^r = \hat{C}_i(s^{r-1}) - (\underline{\eta}\bar{\eta}) \cdot \hat{C}_i(\hat{s}_i^{r-1}, s_{-i}^{r-1})$. Both the strategy \hat{s}_i^{r-1} and the value Δ_i^r are broadcast to all the other players. If $\Delta_i^r \leq 0$ for all $i \in [N]$, then the ABRD stops, and every player i sets $s_i^r = s_i^{r-1}$; in this case, we say that the ABRD *converges*. Otherwise, fix $\Delta^r = \sum_{i \in [N]} \Delta_i^r$ and choose some player $i' \in [N]$ so that $\Delta_{i'}^r > 0$ and $\Delta_{i'}^r \geq \frac{1}{N}\Delta^r$ to update her strategy, setting $s^r = (\hat{s}_{i'}^{r-1}, s_{-i'}^{r-1})$ (the existence of such a player is guaranteed by the pigeonhole principle, and ties are always broken by choosing the player with the smallest index). Such an update can be performed by each

⁴ All subsequent occurrences of the term ABR (and ABRD) share the same approximation parameter $\varrho\underline{\eta}\bar{\eta}$, hence we may refrain from mentioning this parameter explicitly.

player in a distributed manner, as every player has the knowledge of the full vectors $\{s_i^r\}_{i \in [N]}$ and $\{\Delta_i^r\}_{i \in [N]}$. When the ABRD terminates (either because it has reached round $r = R$ or because it converges), **Bayes-ABRD** outputs the strategy generated in the last round.

► **Remark.** Note that **Bayes-ABRD** is designed for computing the strategy profile, not for invoking the strategies to decide the actions in real-time. All the operations of **Bayes-ABRD**, including broadcasting the strategy \hat{s}_i^{r-1} and the value Δ_i^r for every player i in every round $r \in [R]$, are carried out in a “precomputing stage” without seeing the realized type profile. The decision making that happens in real-time does not involve any further communication.

5 Analysis Sketch

Using the property of smoothness parameters, our analysis first gives the upper bound on the BCR with the tuple $(\lambda, \mu, K, \underline{\eta}, \bar{\eta})$ of parameters.⁵

► **Theorem 4.** Let $Q = \frac{2(\underline{\eta}\bar{\eta})N}{1-\varrho(\underline{\eta}\bar{\eta})^2\mu}$. If $R = \lceil Q \cdot \ln(KN^{\alpha_{\max}-1}) \rceil$, then the output s^{out} of **Bayes-ABRD** satisfies $C(s^{out}) \leq \frac{2K\varrho(\underline{\eta}\bar{\eta})^2\lambda}{1-\varrho(\underline{\eta}\bar{\eta})^2\mu} \cdot \mathbb{E}_{t \sim T}[\text{OPT}(t)]$.

Next, we consider the case where the parameters ϱ , $\underline{\eta}$ and $\bar{\eta}$ are fixed, and focus on finding proper smoothness parameters (λ, μ) such that the BGND game is (λ, μ) -smooth with $\mu < 1/[\varrho(\underline{\eta}\bar{\eta})^2]$. For any $\mu' \in (0, \frac{1}{\varrho(\underline{\eta}\bar{\eta})^2})$, define $g_{\mu'}(x) = (x+1)^{\alpha_{\max}-1} - \mu' \cdot x^{\alpha_{\max}}$ and $h(x) = \left[(\alpha_{\max}-1)(x+1)^{\alpha_{\max}-2} \right] / \left[\alpha_{\max} \cdot x^{\alpha_{\max}-1} \right]$. Define $\gamma_{z'}$ to be the unique positive root of $(x+1)^{z'-1} = x^{z'}$ for any $z' \geq 1$ [2]. Let $\mu_\alpha = h(\varrho(\underline{\eta}\bar{\eta})^2 \cdot \gamma_{\alpha_{\max}})$, and $\lambda_\alpha = g_{\mu_\alpha}(\varrho(\underline{\eta}\bar{\eta})^2 \cdot \gamma_{\alpha_{\max}})$. Then we have the following result on the smoothness.

► **Theorem 5.** The BGND game is $(\lambda_\alpha, \mu_\alpha)$ -smooth, and $\varrho(\underline{\eta}\bar{\eta})^2\mu_\alpha < 1 - 1/\alpha_{\max}$.

We then proceed to prove that the BGND game admits a potential function that is K -bounded with $K = \lceil \alpha_{\max} \rceil$.

► **Theorem 6.** For the BGND game, there exists a potential function $\Phi(s)$ that satisfies $\Phi(s) \leq C(s) \leq \lceil \alpha_{\max} \rceil \cdot \Phi(s)$ for any strategy profile s .

Now it remains to compute and analyze the $(\underline{\eta}, \bar{\eta})$ -estimation of the expected cost shares. For any $z \in (0, 1)$ and $z' \geq 1$, define $b_z = ((\beta^\circ)^2 + 1)(1 - \frac{1}{\beta^\circ})^{-z}$ with β° being the unique root of $2\beta^3 - (z+2)\beta^2 - 2 = 0$ in the interval $(1, +\infty)$, and $B_{z'}$ to be the fractional Bell number with the parameter z' [6, 28]. Our analysis utilizes the following propositions.

► **Lemma 7** ([8]). Let $\{X_1, X_2, \dots, X_k, \dots\}$ be a finite set of mutually independent random variables following the Bernoulli distribution supported on $\{0, 1\}$. Then for any $z \geq 1$, $\mathbb{E}\left[\left(\sum_k X_k\right)^z\right] \leq B_z \cdot \max\left\{\mathbb{E}\left[\sum_k X_k\right], \left(\mathbb{E}\left[\sum_k X_k\right]\right)^z\right\}$.

► **Lemma 8.** Let $\{X_1, X_2, \dots, X_k, \dots\}$ be a finite set of Bernoulli random variables that are mutually independent. For any $z' \in (0, 1)$, $\frac{1}{b_{z'}} \leq \mathbb{E}\left[\left(1 + \sum_k X_k\right)^{z'}\right] \leq \left(\mathbb{E}\left[1 + \sum_k X_k\right]\right)^{z'}$.

⁵ The proof of Theorem 4 bears similarity to the analysis in [33, 15]. Hence it is deferred to the attached full version. The main differences between that proof of Theorem [33, 15] are discussed in Sec. 3.

For each action a_i of player i and each resource e , denote the indicator of whether e is contained in a_i by $\delta(a_i, e)$. Formally,

$$\delta(a_i, e) = \begin{cases} 0 & \text{if } e \notin a_i \\ 1 & \text{otherwise} \end{cases}.$$

► **Theorem 9.** For any player i , any edge e , any action a_i , and any strategies s_{-i} , let

$$\widehat{f}_{i,e}(+; s_{-i}) = \sum_{j \in [q]} \xi_{e,j} \left[1 + \sum_{i' \neq i} \sum_{t_{i'} \in T_{i'}} p_{i'}(t_{i'}) \delta(s_{i'}(t_{i'}), e) \right]^{\alpha_j - 1}, \quad (4)$$

then $\frac{\widehat{f}_{i,e}(+; s_{-i})}{\max \left\{ 1, \max_{j: \alpha_j \in (1,2)} b_{\alpha_j - 1} \right\}} \leq f_{i,e}(+; s_{-i}) \leq \widehat{f}_{i,e}(+; s_{-i}) \cdot \left\{ 1, \max_{j: \alpha_j \geq 2} B_{\alpha_j - 1} \right\}$.

Sketch of Proof. Let a_i be an action in A_i satisfying $e \in a_i$. By definition, we have

$$\begin{aligned} f_{i,e}(+; s_{-i}) &= \mathbb{E}_{t_{-i} \sim p_{-i}} [f_{i,e}(a_i, s_{-i}(t_{-i}))] \\ &= \sum_{j \in [N]} \xi_{e,j} \cdot \mathbb{E}_{t_{-i} \sim p_{-i}} [(I_e^{a_i, s_{-i}(t_{-i})})^{\alpha_j - 1}] \\ &= \sum_{j \in [N]} \xi_{e,j} \cdot \mathbb{E}_{t_{-i} \sim p_{-i}} \left[\left(1 + \sum_{i' \in [N]: i' \neq i} \delta(s_{i'}(t_{-i}(i')), e) \right)^{\alpha_j - 1} \right] \\ &= \sum_{j \in [N]} \xi_{e,j} \mathbb{E}_{\{t_{i'} \sim p_{i'}\}_{i' \neq i}} \left[\left(1 + \sum_{i' \neq i} \delta(s_{i'}(t_{i'}), e) \right)^{\alpha_j - 1} \right]. \end{aligned}$$

The last transition holds because the prior distribution p is assumed to be a product distribution.

Now define a finite set of mutually independent Bernoulli random variables $\{X_{i',e}(s)\}_{i' \neq i}$ such that each $X_{i',e}(s)$ takes the value 1 with probability $\sum_{t_{i'}: e \in s_{i'}(t_{i'})} p_{i'}(t_{i'})$. Then it can be inductively proved that

$$\mathbb{E}_{\{t_{i'} \sim p_{i'}\}_{i' \neq i}} \left[\left(1 + \sum_{i' \neq i} \delta(s_{i'}(t_{i'}), e) \right)^{\alpha_j - 1} \right] = \mathbb{E} \left[\left(1 + \sum_{i' \neq i} X_{i',e}(s) \right)^{\alpha_j - 1} \right].$$

Recall that the constant 1 in the last expression above can also be viewed as a Bernoulli random variable which equals to 1 with probability 1. For every $\alpha_j \geq 2$, Lemma 7 gives

$$\begin{aligned} \mathbb{E} \left[\left(1 + \sum_{i' \neq i} X_{i',e}(s) \right)^{\alpha_j - 1} \right] &\leq B_{\alpha_j - 1} \cdot \max \left\{ \mathbb{E} \left[1 + \sum_{i' \neq i} X_{i',e}(s) \right], \left(\mathbb{E} \left[1 + \sum_{i' \neq i} X_{i',e}(s) \right] \right)^{\alpha_j - 1} \right\} \\ &= B_{\alpha_j - 1} \cdot \left(\mathbb{E} \left[1 + \sum_{i' \neq i} X_{i',e}(s) \right] \right)^{\alpha_j - 1}. \end{aligned}$$

The second line holds because $\mathbb{E} \left[1 + \sum_{i' \neq i} X_{i',e}(s) \right] > 1$. Similarly, it can be derived from Lemma 8 that for every $\alpha_j \in (1, 2)$,

$$\mathbb{E} \left[\left(1 + \sum_{i' \neq i} X_{i',e}(s) \right)^{\alpha_j - 1} \right] \leq \left(\mathbb{E} \left[1 + \sum_{i' \neq i} X_{i',e}(s) \right] \right)^{\alpha_j - 1},$$

which also trivially holds for $\alpha_j = 1$. So, $\mathbb{E} \left[\left(1 + \sum_{i' \neq i} X_{i',e}(s) \right)^{\alpha_j - 1} \right] \leq \max \left\{ 1, \max_{j: \alpha_j \geq 2} B_{\alpha_j - 1} \right\} \left(\mathbb{E} \left[1 + \sum_{i' \neq i} X_{i',e}(s) \right] \right)^{\alpha_j - 1}$, and in a similar way, it also be

inferred from Lemma 7 and Lemma 8 that $\mathbb{E}\left[\left(1 + \sum_{i' \neq i} X_{i',e}(s)\right)^{\alpha_j - 1}\right] \geq \left(\mathbb{E}\left[1 + \sum_{i' \neq i} X_{i',e}(s)\right]\right)^{\alpha_j - 1} / \max\{1, \max_{j: \alpha_j < 2} b_{\alpha_j - 1}\}$. Since $\mathbb{E}\left[1 + \sum_{i' \neq i} X_{i',e}(s)\right] = 1 + \sum_{i' \neq i} \sum_{t_{i'}} p_{i'}(t_{i'}) \delta(s_{i'}(t_{i'}), e)$, this proposition holds. \blacktriangleleft

Theorem 9 shows that for any $i \in [N]$, $e \in E$ and any s_{-i} , there exists a $(\max\{1, \max_{\alpha_j \in (1,2)} b_{\alpha_j - 1}\}, \max\{\max_{\alpha_j \geq 2} B_{\alpha_j - 1}, 1\})$ -estimation $\widehat{f}_{i,e}(+; s_{-i})$ of $f_{i,e}(+; s_{-i})$, and the following proposition indicates that such an estimation can be obtained in $\text{poly}(q, N, \{|T_i|\}_{i \in [N]})$ -time.

► **Corollary 10.** *By computing Eq. (4), the desired estimation of each expected cost share is obtained in $O(q \cdot \sum_{i \in [N]} |T_i|)$ -time.*

Plugging Theorem 5, Theorem 6, Theorem 9, and Corollary 10 into Theorem 4 proves our main result, Theorem 2.

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 Sebastian Aland, Dominic Dumrauf, Martin Gairing, Burkhard Monien, and Florian Schoppmann. Exact price of anarchy for polynomial congestion games. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 218–229. Springer, 2006.
- 3 Susanne Albers. Energy-efficient Algorithms. *Commun. ACM*, 53(5):86–96, 2010.
- 4 Noga Alon, Yuval Emek, Michal Feldman, and Moshe Tennenholtz. Bayesian ignorance. *Theoretical Computer Science*, 452:1–11, 2012. Preliminary version appears in Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, pp. 384–391. ACM, 2010.
- 5 Matthew Andrews, Antonio Fernández Anta, Lisa Zhang, and Wenbo Zhao. Routing for Power Minimization in the Speed Scaling Model. *IEEE/ACM Transactions on Networking*, 20(1):285–294, February 2012.
- 6 Evripidis Bampis, Alexander Kononov, Dimitrios Letsios, Giorgio Lucarelli, and Maxim Sviridenko. Energy efficient scheduling and routing via randomized rounding. In *33rd International Conference on Foundations of Software Technology and Theoretical Computer Science*, page 449, 2013.
- 7 Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed Scaling to Manage Energy and Temperature. *J. ACM*, 54(1):3:1–3:39, 2007.
- 8 Daniel Berend and Tamir Tassa. Improved bounds on Bell numbers and on moments of sums of random variables. *Probability and Mathematical Statistics*, 30(2):185–205, 2010.
- 9 Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoss, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *Journal of the ACM (JACM)*, 60(1):6, 2013. Preliminary version in STOC'10.
- 10 Moses Charikar, Chandra Chekuri, To-yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation Algorithms for Directed Steiner Problems. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 192–200, 1998.
- 11 Chandra Chekuri, Guy Even, Anupam Gupta, and Danny Segev. Set Connectivity Problems in Undirected Graphs and the Directed Steiner Network Problem. *ACM Trans. Algorithms*, 7(2):18:1–18:17, 2011.
- 12 Ken Christensen, Pedro Reviriego, Bruce Nordman, Michael Bennett, Mehrgan Mostowfi, and Juan Antonio Maestro. IEEE 802.3 az: the road to energy efficient ethernet. *IEEE Communications Magazine*, 48(11), 2010.

- 13 Teodor Gabriel Crainic, Xiaorui Fu, Michel Gendreau, Walter Rei, and Stein W Wallace. Progressive hedging-based metaheuristics for stochastic network design. *Networks*, 58(2):114–124, 2011.
- 14 E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numer. Math.*, 1(1):269–271, 1959.
- 15 Yuval Emek, Shay Kutten, Ron Lavi, and Yangguang Shi. Approximating Generalized Network Design Under (Dis)Economies of Scale with Applications to Energy Efficiency. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 598–606, New York, NY, USA, 2018. ACM.
- 16 Yuval Emek, Shay Kutten, Ron Lavi, and Yangguang Shi. Bayesian Generalized Network Design. *ArXiv e-prints*, June 2019. [arXiv:1907.00484](https://arxiv.org/abs/1907.00484).
- 17 Matthias Englert and Harald Räcke. Oblivious Routing for the Lp-norm. In *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '09, pages 32–40, Washington, DC, USA, 2009. IEEE Computer Society.
- 18 Matthias Englert and Harald Räcke. Oblivious routing for the Lp-norm. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 32–40. IEEE, 2009.
- 19 Michael L. Fredman and Robert Endre Tarjan. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *J. ACM*, 34(3):596–615, 1987.
- 20 Naveen Garg, Anupam Gupta, Stefano Leonardi, and Piotr Sankowski. Stochastic analyses for online combinatorial optimization problems. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, pages 942–951. Society for Industrial and Applied Mathematics, 2008.
- 21 Anupam Gupta, Mohammad T Hajiaghayi, and Harald Räcke. Oblivious network design. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 970–979. Society for Industrial and Applied Mathematics, 2006.
- 22 Anupam Gupta, R Ravi, and Amitabh Sinha. An edge in time saves nine: LP rounding approximation algorithms for stochastic network design. In *FOCS*, pages 218–227, 2004.
- 23 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*, SODA '08, pages 200–207, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.
- 24 Intel. Enhanced Intel Speedstep Technology for the Intel Pentium M Processor. In *Intel White Paper 301170-001*, 2004.
- 25 Sandy Irani and Kirk R. Pruhs. Algorithmic Problems in Power Management. *SIGACT News*, 36(2):63–76, 2005.
- 26 J.L.W.V. Jensen. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, 30(1):175–193, 1906.
- 27 Gregory Lawler and Hariharan Narayanan. Mixing Times and ϵ -P Bounds for Oblivious Routing. In *Proceedings of the Meeting on Analytic Algorithmics and Combinatorics*, ANALCO '09, pages 66–74, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.
- 28 Konstantin Makarychev and Maxim Sviridenko. Solving Optimization Problems with Dis-economies of Scale via Decoupling. In *55th IEEE Annual Symposium on Foundations of Computer Science*, FOCS, 2014.
- 29 Sergiu Nedeveschi, Lucian Popa, Gianluca Iannaccone, Sylvia Ratnasamy, and David Wetherall. Reducing Network Energy Consumption via Sleeping and Rate-adaptation. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 323–336. USENIX Association, 2008.
- 30 Harald Räcke. Survey on Oblivious Routing Strategies. In *Mathematical Theory and Computational Practice*, volume 5635 of *Lecture Notes in Computer Science*, pages 419–429. Springer Berlin Heidelberg, 2009.

45:16 Bayesian Generalized Network Design

- 31 Ragheb Rahmaniani, Teodor Gabriel Crainic, Michel Gendreau, and Walter Rei. Accelerating the Benders Decomposition Method: Application to Stochastic Network Design Problems. *SIAM Journal on Optimization*, 28(1):875–903, 2018.
- 32 Tim Roughgarden. The price of anarchy in games of incomplete information. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 862–879. ACM, 2012.
- 33 Tim Roughgarden. Intrinsic robustness of the price of anarchy. *Journal of the ACM (JACM)*, 62(5):32, 2015. Preliminary version in STOC’09.
- 34 I Richard Savage. Probability inequalities of the Tehebycheff type. *Journal of Research of the National Bureau of Standards-B. Mathematics and Mathematical Physics B*, 65(3):211–222, 1961.
- 35 Yangguang Shi, Fa Zhang, Jie Wu, and Zhiyong Liu. Randomized oblivious integral routing for minimizing power cost. *Theoretical Computer Science*, 607:221–246, 2015.
- 36 Adam Wierman, Lachlan LH Andrew, and Ao Tang. Power-Aware Speed Scaling in Processor Sharing Systems. In *28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, INFOCOM*, pages 2007–2015, April 2009.
- 37 F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 374–382, 1995.

Obviously Strategyproof Mechanisms for Machine Scheduling

Diodato Ferraioli 

Università degli Studi di Salerno, Italy
dferraioli@unisa.it

Adrian Meier

ETH Zurich, Switzerland
meiera@student.ethz.ch

Paolo Penna

ETH Zurich, Switzerland
paolo.penna@inf.ethz.ch

Carmine Ventre 

King's College London, UK
carmine.ventre@kcl.ac.uk

Abstract

Catering to the incentives of people with limited rationality is a challenging research direction that requires novel paradigms to design mechanisms and approximation algorithms. Obviously strategyproof (OSP) mechanisms have recently emerged as the concept of interest to this research agenda. However, the majority of the literature in the area has either highlighted the shortcomings of OSP or focused on the “right” definition rather than on the construction of these mechanisms.

We here give the first set of *tight* results on the approximation guarantee of OSP mechanisms for scheduling related machines. By extending the well-known cycle monotonicity technique, we are able to concentrate on the algorithmic component of OSP mechanisms and provide some novel paradigms for their design.

2012 ACM Subject Classification Theory of computation → Algorithmic mechanism design

Keywords and phrases Bounded Rationality, Extensive-form Mechanisms, Approximate Mechanism Design

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.46

Related Version A full version of the paper is available at <https://arxiv.org/abs/1805.04190>.

Funding *Diodato Ferraioli*: This author is partially supported by GNCS-INdAM and by the Italian MIUR PRIN 2017 Project ALGADIMAR “Algorithms, Games, and Digital Markets”.

1 Introduction

Mechanism design has been a very active research area that aims to develop algorithms that align the objectives of the designer (e.g., optimality of the solution) with the incentives of self-interested agents (e.g., maximize their own utility). One of the main obstacles to its application in real settings is the assumption of full rationality. Where theory predicts that people should not strategize, lab experiments show that they do (to their own disadvantage): this is, for example, the case for Vickrey’s renown second-price auction; proved to be strategyproof and yet bidders lie when submitting sealed bids. Interestingly, however, lies are less frequent when the very same mechanism is implemented via an ascending auction [18].



© Diodato Ferraioli, Adrian Meier, Paolo Penna, and Carmine Ventre;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 46; pp. 46:1–46:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A vague explanation of this phenomenon is that, from the point of view of a bidder, the strategyproofness of an ascending price auction is *easier to grasp* than the strategyproofness of the second-price sealed bid auction [3]. The key difference is the way these auctions are *implemented*:

- In the second-price sealed-bid auction (direct-revelation implementation), each bidder submits her own bid *once* (either her true valuation or a different value). This mechanism is *strategyproof* meaning that truth-telling is a dominant strategy: *for every report of the other bidders*, the utility when truth-telling is not worse than the utility when bidding untruthfully.
- In the ascending price auction (extensive-form implementation), each bidder is repeatedly offered some price which she can accept (stay in the auction) or reject (leave the auction). In this auction, momentarily *accepting a good price* guarantees a non-negative utility, while *rejecting a good price* or *accepting a bad price* yield non-positive utility. Here good price refers to the private valuation of the bidder and, intuitively, truth-telling in this auction means accepting prices as long as they are not above the true valuation.

Intuitively speaking, in the second type of auction, it is *obvious* for a bidder to decide her strategy, because the utility for the *worst scenario* when truth-telling is at least as good as that of the *best scenario* when cheating. The recent definition of *obviously strategyproof* (OSP) mechanisms [23] formalizes this argument: ascending auctions are OSP mechanisms, while sealed-bid auctions are not. Interestingly, [23] proves that a mechanism is OSP *if and only if* truth-telling is dominant even for bidders who lack contingent reasoning skills.

As being OSP is *stronger* than being strategyproof, it is natural to ask if this has an impact on what can be done by such mechanisms. For instance, the so-called *deferred-acceptance* (DA) auctions [26] are OSP (as they essentially are ascending price auctions), but unfortunately their performance (approximation guarantee) for several optimization problems is quite poor compared to what strategyproof mechanisms can do [9]. Whether this is an inherent limitation of OSP mechanisms or just of this technique is not clear.

One of the reasons behind this open question might be the absence of a general technique for designing OSP mechanisms and the lack of an algorithmic understanding of OSP mechanisms. Specifically, it is well known that strategyproofness is equivalent to certain *monotonicity* conditions of the *algorithm* used by the mechanism for computing the solution (be it an allocation of goods or a path in a network with self-interested agents). Therefore, one can essentially focus on the algorithmic part and study questions regarding the approximation and the complexity. The same type of questions seems much more challenging for OSP mechanisms, as such characterizations are not known. Recent work in the area [5, 27, 24] aims at simplifying the notion of OSP, by looking at versions of the revelation principle for OSP mechanisms. This, for example, allows to think, without loss of generality, at deterministic (rather than randomized) extensive-form mechanisms where each agent moves sequentially (rather than concurrently).

The goal of this work is build the foundations to reason about OSP algorithmically. In particular, we advance the state of the art by providing an algorithmic characterization of OSP mechanisms. Among others, our results show why deferred acceptance auctions [26] – essentially the only known technique to design OSP mechanisms with money – do not fully capture the power of a “generic” OSP mechanism, as the latter may exploit some aspects of the implementation (i.e., extensive-form game) in a crucial way.

Our Contribution. To give an algorithmic characterization of OSP mechanisms, we extend the well known cycle-monotonicity (CMON) technique. This approach allows to abstract the truthfulness of an algorithm in terms of non-negative weight cycles on suitably defined graphs. We show that non-negative weight cycles continue to characterize OSP when the graph of

interest is carefully defined. Our main conceptual contribution is a way to accommodate the OSP constraints, which *depend on the particular extensive-form implementation* of the mechanism, in the machinery of CMON, which is designed to focus on the algorithmic output of mechanism. Interestingly, our technique shows the interplay between algorithms (which solution to return) and how this is implemented as an extensive-form game (what we call the *implementation tree*). Roughly speaking, our characterization says which algorithms can be used for *any* choice of the implementation tree. The ability to choose between different implementation trees is what gives extra power to the designer: for example, the construction of OSP mechanisms based on DA auctions [26] uses always *the same* fixed tree for all problems and instances. Though this yields a simple algorithmic condition, it can be wasteful in terms of optimality (approximation guarantee) as we show herein. In fact, for our results, we will use CMON *two ways* to characterize both algorithmic properties (having fixed an implementation tree) and implementation properties (having fixed the approximation guarantee we want to achieve).

Armed with the OSP CMON technique, we are able to give the first *tight* bounds on the approximation ratio of OSP mechanisms. In particular, we consider the problem of scheduling n related machines (for identical jobs). While the lower bound holds regardless of the size of the domain, the mechanisms that we provide are shown to be OSP only for two- and three-value domains, as we prove that these are the only cases in which non-negative two-cycles are necessary and sufficient.

We show that the optimum for machine scheduling can be implemented OSP-ly when the agents' domains have size two. We prove that given a “balanced” optimum (i.e., a greedy allocation of jobs to machines) we can always find an implementation tree for which OSP is guaranteed. The mechanism directly asks the queried agents to reveal their type; given that the domain only contains two values, this is basically a descending/ascending auction. For domains of size three, instead, we give a lower bound of \sqrt{n} and an essentially tight upper bound of $\lceil\sqrt{n}\rceil$. Interestingly, the latter is proved with two different mechanisms – one assuming more than $\lceil\sqrt{n}\rceil^2$ number of jobs and the second under the hypothesis that there are less than that. On the technical level, these results are shown by using our approach of CMON two ways. We prove that any better than \sqrt{n} -approximate OSP mechanism must have the following structure: for a number of rounds, the mechanism must (i) separate, in its implementation tree, largest and second largest value in the domain; (ii) assign nothing to agents who have maximum value in the domain. The former property restricts the family of implementation trees we can use, whilst the latter restricts the algorithmic output. Our lower bound shows that there is nothing in this intersection. Our matching upper bounds need to find *both* implementation tree and algorithm satisfying OSP and approximation guarantee.

► **Main Theorem** (informal). *The tight approximation guarantee of OSP mechanisms that can be guaranteed over all three-valued domains is \sqrt{n} . The OSP mechanisms use a descending auction (to find the $n - \lceil\sqrt{n}\rceil$ slowest machines) followed by an ascending auction (to find the fastest machine(s)).*

While the general idea of the implementation is that of a descending auction followed by an ascending auction independently of the number of jobs, we need to tailor the design of the mechanisms (namely, their ascending phase) according to the number of jobs to achieve OSP and desired approximation simultaneously. This proves two important points. On one hand, the design of OSP mechanisms is challenging yet interesting as one needs to carefully balance algorithms and their implementation. On the other hand, it proves why fixing the implementation, as in DA auctions, might be the wrong choice. It is indeed straightforward to extend and adapt our analysis in order to prove that any ascending and descending (thus including DA) auction has an approximation of n .

We remark that our mechanisms are, to the best of our knowledge, the first examples of OSP mechanisms with money that do not follow a clock or a posted price auction format (other mechanisms that do not follow these formats have been proposed only for setting without money, namely matching and voting [23, 2, 5, 27]). One of the main messages of our work is exactly that it is possible to combine ascending and descending phases for the implementation trees of algorithms with good approximation guarantees and obtain OSP mechanisms.

Related Works. The notion of OSP mechanism has been introduced recently by [23] and has received a lot of attention in the community. As mentioned above, the class of deferred-acceptance auctions [26] yields OSP mechanisms since every such auction can be implemented as a (suitable) ascending price auction. One of the main advantages of DA auctions is that the construction boils down to the problem of defining a suitable *scoring function* for the bidders [26]. [9] studied the approximability of DA auctions for several optimization problems, and showed that in some cases DA auctions must have an approximation guarantee significantly worse than the best strategyproof mechanism; [9, 19] provide a number of positive results where DA auctions are instead optimal. [15] studies also DA auction for the job scheduling problem: they design an approximate mechanism, but for a different objective function, namely the weighted completion time.

Several works have focused on understanding better the notion of OSP mechanism, and studying settings without money, namely matching and voting. In particular [2, 5, 24] mainly attempt to simplify the notion, whilst [27, 31, 14] define, among other results, stronger and weaker versions of OSP. A couple of recent papers related to ours are [12], where among other settings the authors consider OSP mechanisms with money for machine scheduling, and [21], where this problem is studied in the setting without money. In particular, the lower bound for machine scheduling in [12] is *constant* and uses a particular definition of payments, while here we prove a \sqrt{n} lower bound that follows from the CMON characterization of OSP; their upper bound instead uses monitoring, a model wherein agents pay their reported costs whenever they overbid. Monitoring is also used in [21] to prove an encouraging bound for OSP mechanisms without money and a single task; the bound (asymptotically) matches the performances of strategyproof mechanisms.

Research in algorithmic mechanism design [17, 7] has suggested to focus on “simple” mechanisms to deal with bounded rationality. For example, posted-price mechanisms received huge attention very recently and have been applied to many different settings [4, 11, 1, 10, 8]. In these mechanisms one’s own bid is immaterial for the price paid to get some goods of interest – this should immediately suggest that trying to play the mechanism is worthless no matter the cognitive abilities of the agents. However, posted price mechanisms do not fully capture the concept of simple mechanisms: e.g., ascending price auctions are not posted price mechanisms and still turn out to be “simple” to play and understand.

CMON is a widely used technique in mechanism design that dates back to [28] – a general treatment is given in [25, 16]. This method has been used quite extensively to prove strategyproofness of mechanisms in the classical setting, cf., e.g., [6, 22] and when some form of verification can be adopted, see [30, 20]. Particularly relevant for our work is the research which shows that in order to establish strategyproofness it is sufficient to study cycles of length 2 as in [29].

2 Preliminaries

A mechanism design setting is defined by a set of n *selfish agents* and a set of allowed *outcomes* \mathcal{S} . Each agent i has a *type* $t_i \in D_i$, where D_i is called the *domain* of i . The type t_i is usually assumed to be *private knowledge* of agent i . We will let $t_i(X) \in \mathbb{R}$ denote the *cost* of agent i with type t_i for the outcome $X \in \mathcal{S}$. In our application, we will assume that costs are non-negative; however, our framework and characterization hold in general no matter the sign.

A *mechanism* is a process for selecting an outcome $X \in \mathcal{S}$. To this aim, the mechanism interacts with agents. Specifically, agent i is observed to take *actions* (e.g., saying yes/no) that may depend on her presumed type $b_i \in D_i$ (e.g., saying yes could “signal” that the presumed type has some properties that b_i enjoys). We say that agent i takes *actions compatible with (or according to) b_i* to stress this. We highlight that the presumed type b_i can be different from the real type t_i .

For a mechanism \mathcal{M} , we let $\mathcal{M}(\mathbf{b})$ denote the outcome returned by the mechanism when agents take actions according to their presumed types $\mathbf{b} = (b_1, \dots, b_n)$. In our context, this outcome is given by a pair (f, \mathbf{p}) , where $f = f(\mathbf{b})$ (termed *social choice function* or, simply, algorithm) maps the actions taken by the agents according to \mathbf{b} (i.e., each agent i takes actions compatible with b_i) to a feasible solution in \mathcal{S} , and $\mathbf{p} = \mathbf{p}(\mathbf{b}) = (p_1(\mathbf{b}), \dots, p_n(\mathbf{b})) \in \mathbb{R}^n$ maps the actions taken by the agents according to \mathbf{b} to *payments* from the mechanism to the agents.

Each selfish agent i is equipped with a *utility function* $u_i: D_i \times \mathcal{S} \rightarrow \mathbb{R}$. For $t_i \in D_i$ and for an outcome $X \in \mathcal{S}$ returned by a mechanism \mathcal{M} , $u_i(t_i, X)$ is the utility that agent i has for outcome X when her type is t_i . We define utility as a quasi-linear combination of payments and costs, i.e., $u_i(t_i, \mathcal{M}(b_i, \mathbf{b}_{-i})) = p_i(b_i, \mathbf{b}_{-i}) - t_i(f(b_i, \mathbf{b}_{-i}))$.

A mechanism \mathcal{M} is *strategy-proof* if, for each i , the utility of player i is maximized by playing the extensive-form implementation of \mathcal{M} according to her true type t_i . That is, in a strategy-proof mechanism the actions taken according to the true type are dominant for each agent.

For our application, we will be focusing on *single-parameter* settings, that is, the case in which the private information of each bidder i is a single real number t_i and $t_i(X)$ can be expressed as $t_i w_i(X)$ for some publicly known function w_i . To simplify the notation, we will write $t_i f_i(\mathbf{b})$ when we want to express the cost of a single-parameter agent i of type t_i for the output of social choice function f on input the actions corresponding to a bid vector \mathbf{b} .

Obvious Strategyproofness. We now formally define the concept of obviously strategy-proof deterministic mechanisms. This concept has been introduced in [23]. However, our definition is built on the more accessible ones given by [2] and [12]. As shown in [5, 24], our definition is equivalent to Li’s.¹

Let us first formally model how a mechanism works. An *extensive-form mechanism* \mathcal{M} is defined by a directed tree $\mathcal{T} = (V, E)$, called the *implementation tree*, such that:

- Every leaf ℓ of the tree is labeled with a possible outcome $X(\ell) \in \mathcal{S}$ of the mechanism;
- Every internal vertex $u \in V$ is labeled with an agent $S(u) \in [n]$;

¹ More in detail, our definition of implementation tree is equivalent to the concept of round-table mechanism in [24]. Consequently, our definition of OSP is equivalent to the concept of SP-implementation through a round table mechanism, that is proved to be equivalent to the original definition of OSP for deterministic mechanisms. For a discussion of randomization for OSP mechanisms, we kindly refer the reader to [2, 13].

- Every edge $e = (u, v) \in E$ is labeled with a subset $T(e) \subseteq D = \times_i D_i$ of type profiles such that:
 - The subsets of profiles that label the edges outgoing from the same vertex u are disjoint, i.e., for every triple of vertices u, v, v' such that $(u, v) \in E$ and $(u, v') \in E$, we have that $T(u, v) \cap T(u, v') = \emptyset$;
 - The union of the subsets of profiles that label the edges outgoing from a non-root vertex u is equal to the subset of profiles that label the edge going in u , i.e., $\bigcup_{v: (u,v) \in E} T(u, v) = T(\phi(u), u)$, where $\phi(u)$ is the parent of u in \mathcal{T} ;
 - The union of the subsets of profiles that label the edges outgoing from the root vertex r is equal to the set of all profiles, i.e., $\bigcup_{v: (r,v) \in E} T(r, v) = D$;
 - For every u, v such that $(u, v) \in E$, where u is not the root, and for every two profiles $\mathbf{b}, \mathbf{b}' \in T(\phi(u), u)$ such that $b_i = b'_i$, $i = S(u)$, if \mathbf{b} belongs to $T(u, v)$, then \mathbf{b}' must belong to $T(u, v)$ also.

Roughly speaking, the tree represents the steps of the execution of the mechanism. As long as the current visited vertex u is not a leaf, the mechanism interacts with the agent $S(u)$. Different edges outgoing from vertex u are used for modeling the different actions that the agent $S(u)$ can take during this interaction with the mechanism. In particular, each possible action is assigned to an edge outgoing from u . As suggested above, the action that agent i takes may depend on her presumed type $b_i \in D_i$. That is, different presumed types may correspond to taking different actions, and thus to different edges. The label $T(e)$ on edge $e = (u, v)$ then lists the type profiles that enable the agent $S(u)$ to take those actions that have been assigned to e . In other words, when the agent takes the actions assigned to edge e , then the mechanism (and the other agents) can infer that the type profile must be contained in $T(e)$. The constraints on the edges' label can be then explained as follows: first we can safely assume that different actions must correspond to different type profiles (indeed, if two different actions are enabled by the same profiles we can consider them as a single action); second, we can safely assume that each action must correspond to at least one type profile that has not been excluded yet by actions taken before node u was visited (otherwise, we could have excluded this type profile earlier); third, we have that the action taken by agent $S(u)$ can only inform about her types and not about the type of the remaining agents. The execution ends when we reach a leaf ℓ of the tree. In this case, the mechanism returns the outcome that labels ℓ .

Observe that, according to the definition above, for every profile \mathbf{b} there is only one leaf $\ell = \ell(\mathbf{b})$ such that \mathbf{b} belongs to $T(\phi(\ell), \ell)$. Similarly, to each leaf ℓ there is at least a profile \mathbf{b} that belongs to $T(\phi(\ell), \ell)$. For this reason we say that $\mathcal{M}(\mathbf{b}) = X(\ell)$. Moreover, for every type profile \mathbf{b} and every node $u \in V$, we say that \mathbf{b} is *compatible* with u if $\mathbf{b} \in T(\phi(u), u)$. Finally, two profiles \mathbf{b}, \mathbf{b}' are said to *diverge* at vertex u if there are two vertices v, v' such that $(u, v) \in E$, $(u, v') \in E$ and $\mathbf{b} \in T(u, v)$, whereas $\mathbf{b}' \in T(u, v')$.

For every node u in a mechanism \mathcal{M} such that there are two profiles \mathbf{b}, \mathbf{b}' that diverge at u , we say that u is a *divergent node*, and $i = S(u)$ the corresponding *divergent agent*. For each agent i , we define the *current domain* at node u , denoted $D_i(u)$, such that $D_i(r) = D_i$ for the root r and $D_i(u) = \cup_{\mathbf{b} \in T(\phi(u), u)} b_i$. In words, this is the set of types of i that are compatible with the actions that i took during the execution of the mechanism until node u is reached. Indeed, according to the definition, at each node u in which i diverges, \mathcal{M} partitions $D_i(u)$ in k subsets, where k is the number of children of u , and where for every child v of u , $D_i(v) \subset D_i(u)$ contains the types of bidder i compatible with the action that she takes when interacting with the mechanism at node u .

We are now ready to define obvious strategyproofness. An extensive-form mechanism \mathcal{M} is *obviously strategy-proof (OSP)* if for every agent i with real type t_i , for every vertex u such that $i = S(u)$, for every $\mathbf{b}_{-i}, \mathbf{b}'_{-i}$ (with \mathbf{b}'_{-i} not necessarily different from \mathbf{b}_{-i}), and for every $b_i \in D_i$, with $b_i \neq t_i$, such that (t_i, \mathbf{b}_{-i}) and (b_i, \mathbf{b}'_{-i}) are compatible with u , but diverge at u , it holds that $u_i(t_i, \mathcal{M}(t_i, \mathbf{b}_{-i})) \geq u_i(t_i, \mathcal{M}(b_i, \mathbf{b}'_{-i}))$. Roughly speaking, an obviously strategy-proof mechanism requires that, at each time step agent i is asked to take a decision that depends on her type, the worst utility that she can get if she behaves according to her true type is at least the best utility achievable by behaving differently. We stress that our definition does not restrict the alternative behavior to be consistent with a fixed type. Indeed, as noted above, each leaf of the tree \mathcal{T}_u rooted in u corresponds to a profile $\mathbf{b} = (b_i, \mathbf{b}'_{-i})$ compatible with u : then, our definition implies that the utility of i in the leaves where she plays truthfully is at least as much as the utility in every other leaf of \mathcal{T}_u . Hence, if a mechanism is obviously strategy-proof, then it is also strategy-proof.

We say that an extensive-form mechanism is *trivial* if for every vertex $u \in V$ and for every two type profiles \mathbf{b}, \mathbf{b}' , it holds that \mathbf{b} and \mathbf{b}' do *not* diverge at u . That is, a mechanism is trivial if it never requires agents to take actions that depend on their type. If a mechanism is not trivial, then there is at least one divergent node. On the other hand, every execution of a mechanism (i.e., every path from the root to a leaf in the mechanism implementation tree) may go through at most $\sum_i (|D_i| - 1)$ divergent nodes, the upper bound being the case in which at each divergent node u , the agent $i = S(u)$ separates $D_i(u)$ in $D_i(u) \setminus \{b\}$ and $\{b\}$ for some $b \in D_i(u)$.

Machine Scheduling. Here, we are given a set of m identical jobs to execute and the n agents control related machines. That is, agent i has a job-independent processing time t_i per unit of job (equivalently, an execution speed $1/t_i$ that is independent from the actual jobs). The social choice function f must choose a possible schedule $f(\mathbf{b}) = (f_1(\mathbf{b}), \dots, f_n(\mathbf{b}))$ of jobs to the machines, where $f_i(\mathbf{b})$ denotes the job load assigned to machine i when agents take actions according to \mathbf{b} . The cost that agent i faces for the schedule $f(\mathbf{b})$ is $t_i(f(\mathbf{b})) = t_i \cdot f_i(\mathbf{b})$. We focus on social choice functions f^* minimizing the *makespan*, i.e., $f^*(\mathbf{b}) \in \arg \min_{\mathbf{x}} \max_{i=1}^n b_i(\mathbf{x})$. We say that f is ρ -approximate if it returns a solution whose cost is at most ρ times the optimum.

3 Cycle-monotonicity for OSP Mechanisms

We now show how to generalize the cycle-monotonicity technique to design OSP mechanisms.

Let us consider an extensive-form mechanism $\mathcal{M} = (f, \mathbf{p})$ with implementation tree \mathcal{T} .

► **Definition 1** (separating vertices). *A vertex u in the implementation tree \mathcal{T} is $\alpha\beta$ -separating for agent i if the following holds: Node u is labelled with i , i.e., $i = S(u)$; there are two profiles $(\alpha, \mathbf{a}_{-i})$ and (β, \mathbf{b}_{-i}) which are compatible with u but diverge at u , where $\mathbf{a}_{-i}, \mathbf{b}_{-i} \in D_{-i}(u) = \times_{j \neq i} D_j(u)$.*

Note that there might exist several $\alpha\beta$ -separating vertices for agent i as the agent may be asked to separate α from β in different paths from the root to a leaf (but only once for every such path).

The algorithmic characterization of OSP we provide herein is based on the following observation.

► **Observation 2.** *An extensive-form mechanism $\mathcal{M} = (f, \mathbf{p})$ with implementation tree \mathcal{T} is OSP if and only if for all i , for all $\alpha, \beta \in D_i$, $\alpha \neq \beta$, for all vertices u that are $\alpha\beta$ -separating for i :*

$$p_i(\beta, \mathbf{b}_{-i}) - p_i(\alpha, \mathbf{a}_{-i}) \leq \alpha(f(\beta, \mathbf{b}_{-i})) - \alpha(f(\alpha, \mathbf{a}_{-i})) \quad \text{for all } \mathbf{a}_{-i}, \mathbf{b}_{-i} \in D_{-i}(u). \quad (1)$$

We next restate these conditions in terms of suitable weighted graphs and their cycles.

► **Definition 3** (OSP-graph). *Let f be a social choice function and \mathcal{T} be an implementation tree. We define for every agent i , the OSP-graph $OSP_i^{(f,\mathcal{T})}$ as follows: There is a node for each type profile in D , and a directed edge $e = ((\alpha, \mathbf{a}_{-i}), (\beta, \mathbf{b}_{-i}))$ for every $\alpha, \beta \in D_i$, $\alpha \neq \beta$, and $\mathbf{a}_{-i}, \mathbf{b}_{-i} \in D_{-i}(u)$, where u is an $\alpha\beta$ -separating vertex of \mathcal{T} . The weight of the edge is $w(e) = \alpha(f(\beta, \mathbf{b}_{-i})) - \alpha(f(\alpha, \mathbf{a}_{-i}))$.*

► **Definition 4** (OSP CMON). *We say that the OSP cycle monotonicity (OSP CMON) property holds if, for all i , the graph $OSP_i^{(f,\mathcal{T})}$ does not have negative weight cycles. Moreover, we say that the OSP two-cycle monotonicity (OSP 2CMON) holds if the same is true when considering cycles of length two only, i.e., cycles with two edges only.*

► **Theorem 5.** *A mechanism with implementation tree \mathcal{T} is an OSP mechanism for a social function f on finite domains if and only if OSP CMON holds.*

The proof of the theorem follow standard arguments used for the classical definition of strategyproofness. For our application, it is useful to recast the OSP CMON and OSP 2CMON for the case of single-parameter agents.

► **Proposition 6.** *For single-parameter settings, OSP 2CMON is equivalent to the following condition. For every i , for any $\alpha, \beta \in D_i$ with $\alpha < \beta$, for any $\alpha\beta$ -separating node u of \mathcal{T} , with $i = S(u)$, it holds*

$$f_i(\alpha, \mathbf{a}_{-i}) \geq f_i(\beta, \mathbf{b}_{-i}) \quad \text{for all } \mathbf{a}_{-i}, \mathbf{b}_{-i} \in D_{-i}(u). \quad (2)$$

Warm-up: Using OSP CMON to Bound Approximation Guarantee. We next give a simple lower bound for the machine scheduling problem. This simple result gives a taster of the power of OSP CMON as a tool to answer algorithmic questions about OSP.

► **Proposition 7.** *For the machine scheduling problem, no OSP mechanism can be better than 2-approximate, even for two jobs and two agents with three-value domains $D_i = \{L, M, H\}$, where $L < M < H$, with $M > 3L$ and $H > 3M$.*

Proof. Assume by contradiction that there is an OSP mechanism \mathcal{M} that is better than 2-approximate, and let \mathcal{T} be its implementation tree. Since $M > 3L$ and $H > 3M$, every trivial OSP mechanism must have approximation guarantee at least 2. Hence \mathcal{M} must be non trivial. Let i be the first divergent agent of \mathcal{M} implemented with \mathcal{T} , and let u be the node where this agent diverges (such an agent exists because the mechanism is not trivial). We show that this mechanism cannot satisfy OSP 2CMON, thus a contradiction.

If i diverges at u on M and H , then consider $\mathbf{b} = (\beta, \mathbf{b}_{-i}) = (H, H)$ and $\mathbf{a} = (\alpha, \mathbf{a}_{-i}) = (M, L)$. Since the mechanism is better than 2-approximate, it must satisfy $f_i(\beta, \mathbf{b}_{-i}) = 1$ and $f_i(\alpha, \mathbf{a}_{-i}) = 0$. Note that this violates the OSP 2CMON condition (Equation 2 in Proposition 6): Since i is the first divergent agent, and u is the corresponding node, the set $D_{-i}(u)$ consists of all types in the domain of the other agent, and therefore $H, L \in D_{-i}(u)$ as required to invoke (2) with our choice $\mathbf{b}_{-i} = H$ and $\mathbf{a}_{-i} = L$. If i diverges at u on L and M , then consider $\mathbf{a} = (\alpha, \mathbf{a}_{-i}) = (L, L)$ and $\mathbf{b} = (\beta, \mathbf{b}_{-i}) = (M, H)$. Since the mechanism is better than 2-approximate, it must satisfy $f_i(\alpha, \mathbf{a}_{-i}) = 1$ and $f_i(\beta, \mathbf{b}_{-i}) = 2$. Similarly to the previous case, this violates the OSP 2CMON condition (2). ◀

Note that for this bound we require the domain to have at least three different values; we will in fact prove in Section 4 that we can design an optimal OSP mechanism for scheduling related machines when $D_i = \{L_i, H_i\}$ for every i . We will also show how to use a more involved argument to prove a substantially higher (and tight) bound of \sqrt{n} .

Two-cycles are Sufficient for Single-parameter Domains of Size at most Three. Two-cycle monotonicity is a property easier to work with than CMON. We will now observe that, for single parameter settings, these properties turns out to be equivalent if and only if $D_i = \{L_i, M_i, H_i\}$ for each i , with $L_i \leq M_i \leq H_i$.

► **Theorem 8.** *Consider a single-parameter setting where $|D_i| \leq 3$ for each agent i . A mechanism with implementation tree \mathcal{T} and social choice function f is OSP iff OSP 2CMON holds.*

We next show that this result is essentially tight in the sense that OSP 2CMON does not imply OSP CMON (and thus OSP-ness) already in four-value domains.

► **Theorem 9.** *There exists a mechanism for which OSP 2CMON holds for every agent, but there is an agent i for which the mechanism does not satisfy OSP CMON, whenever $|D_i| \geq 4$. The claim holds even for a single-item auction setting and $D_j = D$ for every $j \neq i$.*

4 Scheduling Related Machines

In this section, we show how the domain structure impacts on the performance guarantee of OSP mechanisms, for the problem of scheduling related machines. Roughly speaking, the problem is easy for *two-value* domains, while it becomes difficult already for *three-value* domains and *two* jobs.

We can prove that an OSP optimal mechanism exists for the case in which each agent's domain has size two. Specifically, we have the following theorem.

► **Theorem 10.** *For the machine scheduling problem, there exists an optimal polynomial-time OSP mechanism for any number of agents with two-value domains $D_i = \{L_i, H_i\}$.*

Lower Bound for Three-value Domain

We now show how to strengthen Proposition 7 and prove a \sqrt{n} -inapproximability result for three-value domains.

► **Theorem 11.** *For the machine scheduling problem, no OSP mechanism can be better than \sqrt{n} -approximate. This also holds for three-value domains $D_i = \{L, M, H\}$.*

For the proof, we consider $m = n = c^2$, for some $c > 1$, and a three-value domain $D_i = \{L, M, H\}$ such that $M \geq m \cdot L$ and $H \geq m\sqrt{n} \cdot M$. Observe that, in such domains, every trivial mechanism must have an approximation ratio not lower than \sqrt{n} . Consider then a non-trivial mechanism \mathcal{M} and let \mathcal{T} be its implementation tree. Let us rename the agents as follows: Agent 1 is the 1st agent that diverges in \mathcal{T} ; since the mechanism is not trivial agent 1 exists. We now call agent 2, the 2nd distinct agent that diverges in the subtree of \mathcal{T} defined by agent 1 taking an action compatible with type H ; if no agent diverges in this subtree of \mathcal{T} we simply call 2 an arbitrary agent different from 1. More generally, agent i is the i th distinct agent that diverges, if any, in the subtree of \mathcal{T} that corresponds to the case that the actions previously taken by agents are compatible with their type being H . As above, if no agent diverges in the subtree of interest, we just let i denote an arbitrary agent different from $1, 2, \dots, i - 1$. We denote with u_i the node in which i first diverges in the subtree in which all the other agents have taken actions compatible with H ; if i does not diverge (i.e., got her id arbitrarily) we denote with u_i a dummy node in which we will say that i does not diverge and i takes an action compatible with every type in D_i . We then have the following lemma.

► **Lemma 12.** Any OSP mechanism \mathcal{M} which is k -approximate, with $k < \sqrt{n}$, must satisfy:

1. For every $i \leq n - \sqrt{n} + 1$, if agent i diverges at node u_i , it must diverge on M and H .
2. For every $i \leq n - \sqrt{n}$, if agent i diverges at node u_i and takes an action compatible with her type being H , then \mathcal{M} does not assign any job to i , regardless of the actions taken by the other agents.

Proof. Let us first prove part (1). Suppose that there is $i \leq n - \sqrt{n} + 1$ such that at node u_i i diverges on L and $\{M, H\}$. Consider the type profile \mathbf{x} such that $x_i = M$, and $x_j = H$ for every $j \neq i$. Observe that \mathbf{x} is compatible with node u_i . The optimal allocation for the type profile \mathbf{x} assigns all jobs to machine i , with cost $OPT(\mathbf{x}) = mM$. Since \mathcal{M} is k -approximate, then it also assigns all jobs to machine i . Indeed, if a job is assigned to a machine $j \neq i$, then the cost of the mechanism would be at least $H \geq \sqrt{n} \cdot mM > k \cdot OPT(\mathbf{x})$, that contradicts the approximation bound.

Consider now the profile \mathbf{y} such that $y_i = L$, $y_j = H$ for every $j < i$, and $y_j = L$ for every $j > i$. Observe that also \mathbf{y} is compatible with node u_i . It is not hard to see that $OPT(\mathbf{y}) \leq \left\lceil \frac{m}{n-i+1} \right\rceil \cdot L$. Since \mathcal{M} is k -approximate, then it cannot assign all jobs to machine i . Indeed, in this case the cost of the mechanism contradicts the approximation bound, since it would be $mL \geq \sqrt{n} \left\lceil \frac{m}{n-i+1} \right\rceil L > k \cdot OPT(\mathbf{y})$, where we used that $\sqrt{n} \left\lceil \frac{m}{n-i+1} \right\rceil \leq \sqrt{n} \left\lceil \frac{m}{\sqrt{n}} \right\rceil = \sqrt{n} \left\lceil \frac{m}{\sqrt{n}} \right\rceil = \sqrt{n} \cdot \sqrt{n} = n = m$.

Hence, we have that if i takes actions compatible with M , then there exists a type profile compatible with u_i such that i receives n jobs, whereas, if i takes a different action compatible with a lower type, then there exists a type profile compatible with u_i such that i receives less than n jobs. However, this contradicts the OSP CMON property.

Let us now prove part (2). Suppose that there is $i \leq n - \sqrt{n}$ and \mathbf{x}_{-i} compatible with u_i such that if i takes an action compatible with type H , then \mathcal{M} assigns at least a job to i . According to part (1), machine i diverges at node u_i on H and M .

Consider then the profile \mathbf{y} such that $y_i = M$, $y_j = H$ for $j < i$, and $y_j = L$ for $j > i$. It is easy to see that the optimal allocation has cost $OPT(\mathbf{y}) = \left\lceil \frac{m}{n-i} \right\rceil \cdot L$. Since \mathcal{M} is k -approximate, then it does not assign any job to machine i . Otherwise, the mechanism contradicts the approximation bound since his cost would be at least $M \geq mL \geq \sqrt{n} \left\lceil \frac{m}{n-i} \right\rceil L > k \cdot OPT(\mathbf{x})$, where we used that $\sqrt{n} \left\lceil \frac{m}{n-i} \right\rceil \leq \sqrt{n} \left\lceil \frac{m}{\sqrt{n}} \right\rceil = \sqrt{n} \left\lceil \frac{m}{\sqrt{n}} \right\rceil = \sqrt{n} \cdot \sqrt{n} = n = m$.

Hence, we have that if i takes actions compatible with H , then there exists a type profile compatible with u_i such that i receives one job, whereas, if i takes a different action compatible with a lower type, then there exists a type profile compatible with u_i such that i receives zero jobs. However, this contradicts the OSP CMON property. ◀

Proof of Theorem 11. Suppose that there is an OSP k -approximate mechanism \mathcal{M} for some $k < \sqrt{n}$, thus implying that the mechanism is not trivial.

Assume first that for all $i \leq n - \sqrt{n}$ agent i diverges at u_i . Consider \mathbf{x} such that $x_i = H$ for every i . Observe that \mathbf{x} is compatible with u_i for every i . The optimal allocation consists in assigning a job to each machine, and has cost $OPT(\mathbf{x}) = H$. According to Part (2) of Lemma 12, if machines take actions compatible with \mathbf{x} , then the mechanism \mathcal{M} does not assign any job to machine i , for every $i \leq n - \sqrt{n}$. Hence, the best outcome that \mathcal{M} can return for \mathbf{x} consists in assigning \sqrt{n} jobs to each of the other \sqrt{n} machines. Therefore, the cost of \mathcal{M} is at least $\sqrt{n}H > kOPT(\mathbf{x})$, which contradicts the approximation ratio of \mathcal{M} .

Consider now the case that there is $1 < i \leq n - \sqrt{n}$ that does not diverge at u_i (since the mechanism is not trivial $i > 1$). This means that all the machines $j \geq i$ will not diverge at u_i ; let S denote this set of machines. Note that the $n - i + 1 \geq \sqrt{n} + 1$ machines in S will have

the same outcome no matter their types when the machines not in S have type H ; in other words, any profile \mathbf{x} where $x_j = H$ for $j \notin S$ is compatible with u_i . Consider \mathbf{x} such that $x_j = H$ for $j \notin S$ and $x_j = L$ otherwise. Since $H \geq n^{5/2}L$, to guarantee approximation k , the mechanism must return a solution for \mathbf{x} which keeps the machines not in S empty; then there is a $j^* \in S$ which is allocated at least $\lceil \frac{n}{\sqrt{n+1}} \rceil$ jobs. Consider now \mathbf{y} where $y_j = H$ for $j \notin S \setminus \{j^*\}$ and $y_j = L$ otherwise. The mechanism must give in output the same allocation given in output for \mathbf{x} since it cannot distinguish \mathbf{x} from \mathbf{y} . However, giving that many jobs to machine j^* contradicts the approximation guarantee on \mathbf{y} . ◀

The arguments above can be used to prove that ascending and descending auctions do not help in this setting. Specifically, they cannot return an approximation better than n .

Upper bound for Three-value Domain

We describe our mechanisms for a generic domain, as this turns out to be useful in the analysis. In what follows, the usual bold notation \mathbf{x} denotes vectors of n entries, while a “hat-bold” notation $\hat{\mathbf{x}}$ denotes vectors of $\lceil \sqrt{n} \rceil$ entries only.

A Mechanism for Many Jobs (Large m). We now introduce mechanism \mathcal{M}_{many} whose approximation ratio approaches $\lceil \sqrt{n} \rceil$, whenever $m \gg \lceil \sqrt{n} \rceil$. The mechanism consists of a descending Phase (Algorithm 1) followed by an ascending Phase (Algorithm 2). The descending phase simply queries the agents to identify (and forget about) the $n - \sqrt{n}$ slowest machines; the ascending phase instead identifies the fastest machine and then computes the optimal solution to a vector where the types of the remaining $\sqrt{n} - 1$ machines is set to the best type of the slow machines found in the descending phase.

■ **Algorithm 1** Descending Phase (for both mechanisms \mathcal{M}_{many} and \mathcal{M}_{few}).

```

1 Set  $A = [n]$ , and  $t_i = \max\{d \in D_i\}$ 
2 while  $|A| > \lceil \sqrt{n} \rceil$  do
3   Set  $p = \max_{a \in A}\{t_a\}$  and  $i = \min\{a \in A : t_a = p\}$ 
4   Ask machine  $i$  if her type is equal to  $p$ 
5   if yes then remove  $i$  from  $A$ 
6   else set  $t_i = \max\{t \in D_i : t < p\}$ 

```

► **Proposition 13.** *Mechanism \mathcal{M}_{many} is OSP for any three-value domain $D_i = \{L_i, M_i, H_i\}$.*

Proof. We prove that \mathcal{M}_{many} satisfies OSP 2CMON. The claim then follows from Theorem 8. Specifically, for each machine i , for each node u in which the mechanism makes a query to i , for each pair of type profiles \mathbf{x}, \mathbf{y} compatible with u such that i diverges at u between x_i and y_i , we need to prove that if $x_i > y_i$, then $f_i(\mathcal{M}_{many}(\mathbf{x})) \leq f_i(\mathcal{M}_{many}(\mathbf{y}))$.

Let us first consider a node u corresponding to the descending phase of the mechanism. In this case, $x_i = p$, where p is as at node u . Moreover, in all profiles compatible with u there are at least $\lceil \sqrt{n} \rceil$ machines that either have a type lower than p , or they have type p but are queried after i . However, for every \mathbf{x}_{-i} satisfying this property, we have that $f_i(\mathcal{M}_{many}(\mathbf{x})) = 0$, which implies that these two-cycles have non-negative weight.

Suppose now that node u corresponds to the ascending phase of the mechanism. In this case, $y_i = p$, where p is as at node u . Observe that for every \mathbf{y}_{-i} compatible with node u , $f_i(\mathcal{M}_{many}(\mathbf{y})) = f_i^*(y_i, \hat{\mathbf{z}}_{-i})$, where $f_i^*(y_i, \hat{\mathbf{z}}_{-i})$ is the number of jobs assigned to machine i by the optimal outcome on input profile $(y_i, \hat{\mathbf{z}}_{-i})$, $\hat{\mathbf{z}}_{-i}$ being such that $\hat{z}_j = \max_{k \in A} t_k$

for every $j \in A \setminus \{i\}$. Observe that for every \mathbf{x} compatible with u , it must be the case that $x_j \geq y_i$ for every $j \in A$. Hence, we can distinguish two cases: if $\min_j x_j = x_i$, then $f_i(\mathcal{M}_{many}(\mathbf{x})) = f_i^*(x_i, \hat{\mathbf{z}}_{-i}) \leq f_i^*(y_i, \hat{\mathbf{z}}_{-i}) = f_i(\mathcal{M}_{many}(\mathbf{y}))$; if instead $\min_j x_j = x_k$, for some $k \neq i$, then $f_i(\mathcal{M}_{many}(\mathbf{x})) = f_i^*(x_k, \hat{\mathbf{z}}_{-k}) \leq f_k^*(x_k, \hat{\mathbf{z}}_{-k}) \leq f_i^*(y_i, \hat{\mathbf{z}}_{-i}) = f_i(\mathcal{M}_{many}(\mathbf{y}))$, where we used that $\hat{\mathbf{z}}_{-k} = \hat{\mathbf{z}}_{-i}$ and the inequalities follow since: (i) in the optimal outcome the fastest machine must receive at least as many jobs as slower machines; (ii) the optimal outcome is monotone, (i.e., given the speeds of other machines, the number of jobs assigned to machine i decreases as its speeds decreases). ◀

■ **Algorithm 2** Ascending Phase (\mathcal{M}_{many}).

```

1 Set  $s_i = \min\{d \in D_i\}$ 
2 while  $|A| > 0$  do
3   Set  $p = \min_{a \in A}\{s_a\}$  and
    $i = \min\{a \in A : s_a = p\}$ 
4   Ask machine  $i$  if her type is  $p$ 
5   if yes then
6     Let  $\hat{\mathbf{z}}$  be s.t.  $\hat{z}_i = p$  and
      $\hat{z}_j = \min_{k \notin A} t_k$  for  $j \in A, j \neq i$ 
7     Let  $f^*(\hat{\mathbf{z}}) = (f_i^*(\hat{\mathbf{z}}))_{i \in A}$  be the
     optimal assignment for profile  $\hat{\mathbf{z}}$ 
8     Assign  $f_j^*(\hat{\mathbf{z}})$  jobs to each  $j \in A$ 
9     Set  $A = \emptyset$ 
10  else set  $s_i = \min\{d \in D_i : d > p\}$ 

```

■ **Algorithm 3** Ascending Phase (\mathcal{M}_{few}).

```

1 Set  $t_a = \min_i\{d \in D_i\}$  and  $C = m$ 
2 while  $|A| > 0$  do
3   Set  $q = \min_{a \in A}\{t_a\}$  and
    $i = \min\{a \in A : t_a = q\}$ 
4   Ask machine  $i$  if her type is  $q$ 
5   if yes then
6     Let  $\zeta = \lceil C/|A| \rceil$ 
7     Let  $z$  be the largest integer in
      $[\zeta, C]$  such that  $z \cdot q \leq \lceil \sqrt{n} \rceil \cdot p$ 
8     Assign  $z$  jobs to  $i$ 
9     Set  $C = C - z$ 
10    Remove  $i$  from  $A$ 
11  else set  $t_i = \min\{d \in D_i : d > p\}$ 

```

The next theorem bounds the approximation ratio of the mechanism

► **Theorem 14.** *Mechanism \mathcal{M}_{many} is $(\lceil \sqrt{n} \rceil + 1)$ -approximate for $m > \lceil \sqrt{n} \rceil^2$.*

A Mechanism for Few Jobs (Small m). We now introduce a mechanism \mathcal{M}_{few} which is OSP and $\lceil \sqrt{n} \rceil$ -approximate whenever $m \leq \lceil \sqrt{n} \rceil^2$. Like \mathcal{M}_{many} , \mathcal{M}_{few} consists of a descending phase followed by an ascending phase. The descending phase is exactly the same (Algorithm 1) with the difference that the ascending phase (Algorithm 3) does not need the information on the type of the machines that are not in A at that point.

We show next that \mathcal{M}_{few} is well defined under our assumption on m , is OSP and has approximation $\lceil \sqrt{n} \rceil$.

► **Lemma 15.** *If $m \leq \lceil \sqrt{n} \rceil^2$ then there exists a z in line 7 of Algorithm 3.*

Proof. We next show that it never occurs during the ascending phase that $\zeta \cdot q > \lceil \sqrt{n} \rceil \cdot p$. Indeed, for the first machine to reveal the type during the ascending phase, we have that $|P_0| = m \leq \lceil \sqrt{n} \rceil^2$, and, thus $\zeta \leq \lceil \sqrt{n} \rceil$. Hence, $\zeta \cdot q \leq \lceil \sqrt{n} \rceil \cdot p$ since $q \leq p$. If a set $Q \subset A$ of machines has previously revealed the type during the ascending phase, and the execution of this phase has not been stopped, then these machines received at least $m' = \lfloor |Q|m/|A| \rfloor + \min\{|Q|, m \bmod |A|\}$ jobs. Then $|P_0| = m - m' \leq (|A| - |Q|) \lceil \sqrt{n} \rceil$, and thus $\zeta \leq \lceil \sqrt{n} \rceil$, and, since $q \leq p$, $\zeta \cdot q \leq \lceil \sqrt{n} \rceil \cdot p$. ◀

► **Proposition 16.** *Mechanism \mathcal{M}_{few} is OSP for three-value domains $D_i = \{L_i, M_i, H_i\}$.*

Proof. We prove that \mathcal{M}_{many} satisfies the OSP 2CMON. The claim then follows from Theorem 8. Specifically, for each machine i , for each node u in which the mechanism makes a query to i , for each pair of type profiles \mathbf{x}, \mathbf{y} compatible with u such that x_i and y_i diverge at u , we need to prove that if $x_i > y_i$, then $f_i(\mathcal{M}_{few}(\mathbf{x})) \leq f_i(\mathcal{M}_{few}(\mathbf{y}))$.

Let us first consider a node u corresponding to the descending phase of the mechanism. In this case, $x_i = p$, where p is as at node u . Moreover, in all profiles compatible with u there are at least $\lceil \sqrt{n} \rceil$ machines that either have a type lower than p , or they have type p but they are queried after i . Hence, for every \mathbf{x}_{-i} satisfying this property, we have that $f_i(\mathcal{M}_{few}(\mathbf{x})) = 0$, that implies the claim.

Suppose now that node u corresponds to the ascending phase of the mechanism. Let $C(u)$, $A(u)$, $p(u)$ and $q(u)$ be the value of C , A , p and q at that node. Observe that for every profile compatible with u , the type of machines not in $A(u)$ is fixed, whereas for every machine in $A(u)$, the type is at least $q(u)$. Moreover, $y_i = q(u)$. Hence, $f_i(\mathcal{M}_{few}(\mathbf{y}))$ is the largest integer $z \leq C(u)$ such that $z \cdot y_i \leq \lceil \sqrt{n} \rceil \cdot p(u)$. On the other side, for every $x_i > y_i$, $f_i(\mathcal{M}_{few}(\mathbf{x}))$ is at most the largest integer $z' \leq C(u)$ such that $z' \cdot x_i \leq \lceil \sqrt{n} \rceil \cdot p(u)$. Since $x_i > y_i$, then $z' \leq z$, and the lemma follows. \blacktriangleleft

► **Proposition 17.** *Mechanism \mathcal{M}_{few} is $\lceil \sqrt{n} \rceil$ -approximate.*

5 Conclusions

We have focused on OSP mechanisms, a compelling and needed notion of incentive compatibility for bounded rationality; [23] proves that OSP is the notion that captures strategyproofness for agents who lack contingent reasoning skills. It is thus paramount to understand the limitations and the power of these mechanisms.

We have introduced a new technique to look at OSP mechanisms, and shown its power by giving tight results on the approximation for a paradigmatic problem in the area. Our contribution highlights how there are two dimensions, algorithms and their implementation, to the design of these mechanisms. The interplay between these dimensions is encapsulated by OSP CMON and plays a central role, as shown by the limitations of fixing the implementation beforehand (as in DA auctions or direct revelation mechanisms).

Furthermore, the significance of the technique can be seen by comparing the previously known lower bounds on the approximation guarantee of OSP mechanisms given in [12, 5]. These results focus on the first divergent agent only and bound the *strategyproof* payments for the identified instances in order to understand and limit the behavior of the algorithm. As a result, their bounds are small constants (2 for machine scheduling in [12] and $1 + \epsilon$, for combinatorial auctions with additive bidders in [5]).

We leave a number of open problems. A technical one is about the domain size and the difference between 2-cycles and longer ones; to what extent adding an extra type in the domain can deteriorate the approximation ratio of OSP mechanisms? A second, more conceptual question, is about dealing with multi-parameter agents. Even with the machinery of OSP CMON, it does not seem immediate to characterize the implementation trees for this kind of agents as there is not a concept of relative ordering of types. Hence, the common pattern of OSP mechanisms, where at each node of the implementation tree an extreme of the current domain is separated from the rest, cannot be adopted.

References

- 1 M. Adamczyk, A. Borodin, D. Ferraioli, B. de Keijzer, and S. Leonardi. Sequential Posted Price Mechanisms with Correlated Valuations. In *WINE 2015*, pages 1–15, 2015.
- 2 I. Ashlagi and Y. A. Gonczarowski. Stable Matching Mechanisms are not Obviously Strategy-proof. *J. Economic Theory*, 177:405–425, 2018.
- 3 L. M. Ausubel. An Efficient Ascending-bid Auction for Multiple Objects. *American Economic Review*, 94(5):1452–1475, 2004.
- 4 M. Babaioff, N. Immorlica, B. Lucier, and S. M. Weinberg. A Simple and Approximately Optimal Mechanism for an Additive Buyer. In *FOCS 2014*, pages 21–30, 2014.
- 5 S. Bade and Y. A. Gonczarowski. Gibbard-Satterthwaite success stories and obvious strategyproofness. In *EC 2017*, page 565, 2017.
- 6 S. Bikhchandani, S. Chatterji, R. Lavi, A. Muálem, N. Nisan, and A. Sen. Weak Monotonicity Characterizes Deterministic Dominant-Strategy Implementation. *Econometrica*, 74(4):1109–1132, 2006.
- 7 S. Chawla, J. Hartline, D. Malec, and B. Sivan. Multi-parameter Mechanism Design and Sequential Posted Pricing. In *STOC 2010*, pages 311–320, 2010.
- 8 J. Correa, P. Foncea, R. Hoeksma, T. Oosterwijk, and T. Vredeveld. Posted Price Mechanisms for a Random Stream of Customers. In *EC 2017*, pages 169–186, 2017.
- 9 P. Dütting, V. Gkatzelis, and T. Roughgarden. The Performance of Deferred-Acceptance Auctions. *Math. Oper. Res.*, 42(4), 2017.
- 10 A. Eden, M. Feldman, O. Friedler, I. Talgam-Cohen, and S. M. Weinberg. A Simple and Approximately Optimal Mechanism for a Buyer with Complements. In *EC 2017*, pages 323–323, 2017.
- 11 M. Feldman, A. Fiat, and A. Roytman. Makespan Minimization via Posted Prices. In *EC 2017*, pages 405–422, 2017.
- 12 D. Ferraioli and C. Ventre. Obvious Strategyproofness Needs Monitoring for Good Approximations. In *AAAI 2017*, pages 516–522, 2017.
- 13 D. Ferraioli and C. Ventre. Probabilistic Verification for Obviously Strategyproof Mechanisms. In *IJCAI 2018*, 2018.
- 14 D. Ferraioli and C. Ventre. Obvious Strategyproofness, Bounded Rationality and Approximation: The Case of Machine Scheduling. In *SAGT 2019*, 2019.
- 15 V. Gkatzelis, E. Markakis, and T. Roughgarden. Deferred-acceptance auctions for multiple levels of service. In *EC 2017*, pages 21–38, 2017.
- 16 H. Gui, R. Müller, and R. V. Vohra. Dominant Strategy Mechanisms with Multidimensional Types. Discussion paper 1392, Northwestern Univ., 2004.
- 17 J. Hartline and T. Roughgarden. Simple versus Optimal Mechanisms. In *EC 2009*, pages 225–234, 2009.
- 18 J. Kagel, R. Harstad, and D. Levin. Information Impact and Allocation Rules in Auctions with Affiliated Private Values: A Laboratory Study. *Econometrica*, pages 1275–1304, 1987.
- 19 A. Kim. Welfare Maximization with Deferred Acceptance Auctions in Reallocation Problems. In *ESA 2015*, pages 804–815, 2015.
- 20 P. Krysta and C. Ventre. Combinatorial Auctions with Verification are Tractable. *Theor. Comput. Sci.*, 571:21–35, 2015.
- 21 M. Kyropoulou and C. Ventre. Obviously Strategyproof Mechanisms without Money for Scheduling. In *AAMAS 2019*, 2019.
- 22 R. Lavi and C. Swamy. Truthful Mechanism Design for Multi-dimensional Scheduling via Cycle Monotonicity. *Games and Economic Behavior*, 67(1):99–124, 2009.
- 23 S. Li. Obviously Strategy-proof Mechanisms. *American Economic Review*, 107(11):3257–87, 2017.
- 24 A. Mackenzie. A Revelation Principle for Obviously Strategy-proof Implementation. Research Memorandum 014, (GSBE), 2017.

- 25 A. Malakhov and R. V. Vohra. Single and Multi-Dimensional Optimal Auctions - A Network Approach. Discussion paper 1397, Northwestern Univ., 2004.
- 26 P. Milgrom and I. Segal. Deferred-acceptance Auctions and Radio Spectrum Reallocation. In *EC 2014*, 2014.
- 27 M. Pycia and P. Troyan. Obvious Dominance and Random Priority. In *EC 2019*, 2019.
- 28 J.-C. Rochet. The Taxation Principle and Multitime Hamilton-Jacobi Equations. *Journal of Mathematical Economics*, 14(2):113–128, 1985.
- 29 M. Saks and L. Yu. Weak Monotonicity Suffices for Truthfulness on Convex Domains. In *EC 2005*, pages 286–293, 2005.
- 30 C. Ventre. Truthful Optimization Using Mechanisms with Verification. *Theor. Comput. Sci.*, 518:64–79, 2014.
- 31 L. Zhang and D. Levin. Bounded rationality and robust mechanism design: An axiomatic approach. *American Economic Review*, 107(5):235–39, 2017.

Going Far From Degeneracy

Fedor V. Fomin

Department of Informatics, University of Bergen, Norway
fomin@ii.uib.no

Petr A. Golovach

Department of Informatics, University of Bergen, Norway
Petr.Golovach@uib.no

Daniel Lokshtanov

Department of Computer Science, University of California Santa Barbara, USA
daniello@ucsb.edu

Fahad Panolan

Department of Computer Science and Engineering, IIT Hyderabad, India
fahad@iith.ac.in

Saket Saurabh

The Institute of Mathematical Sciences, HBNI, Chennai, India
saket@imsc.res.in

Meirav Zehavi

Ben-Gurion University, Beersheba, Israel
meiravze@bgu.ac.il

Abstract

An undirected graph G is d -degenerate if every subgraph of G has a vertex of degree at most d . By the classical theorem of Erdős and Gallai from 1959, every graph of degeneracy $d > 1$ contains a cycle of length at least $d + 1$. The proof of Erdős and Gallai is constructive and can be turned into a polynomial time algorithm constructing a cycle of length at least $d + 1$. But can we decide in polynomial time whether a graph contains a cycle of length at least $d + 2$? An easy reduction from HAMILTONIAN CYCLE provides a negative answer to this question: Deciding whether a graph has a cycle of length at least $d + 2$ is NP-complete. Surprisingly, the complexity of the problem changes drastically when the input graph is 2-connected. In this case we prove that deciding whether G contains a cycle of length at least $d + k$ can be done in time $2^{\mathcal{O}(k)}|V(G)|^{\mathcal{O}(1)}$. In other words, deciding whether a 2-connected n -vertex G contains a cycle of length at least $d + \log n$ can be done in polynomial time. Similar algorithmic results hold for long paths in graphs. We observe that deciding whether a graph has a path of length at least $d + 1$ is NP-complete. However, we prove that if graph G is connected, then deciding whether G contains a path of length at least $d + k$ can be done in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$. We complement these results by showing that the choice of degeneracy as the “above guarantee parameterization” is optimal in the following sense: For any $\varepsilon > 0$ it is NP-complete to decide whether a connected (2-connected) graph of degeneracy d has a path (cycle) of length at least $(1 + \varepsilon)d$.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Longest path, longest cycle, fixed-parameter tractability, above guarantee parameterization

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.47

Related Version The full version of the paper is available at <https://arxiv.org/abs/1902.02526>.

Funding The research leading to these results has received funding from the Research Council of Norway via the projects “CLASSIS” and “MULTIVAL” and from the European Research Council (ERC) via grant LOPPRE, reference 819416.



© Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi;

licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 47; pp. 47:1–47:14



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We thank Nikolay Karpov for communicating to us the question of finding a path above the degeneracy bound and Proposition 15.

1 Introduction

The classical theorem of Erdős and Gallai [11] says that

► **Theorem 1** (Erdős and Gallai [11]). *Every graph with n vertices and more than $(n - 1)\ell/2$ edges ($\ell \geq 2$) contains a cycle of length at least $\ell + 1$.*

Recall that a graph G is d -degenerate if every subgraph H of G has a vertex of degree at most d , that is, the minimum degree $\delta(H) \leq d$. Respectively, the *degeneracy* of graph G , is $\text{dg}(G) = \max\{\delta(H) \mid H \text{ is a subgraph of } G\}$. Since a graph of degeneracy d has a subgraph H with at least $d \cdot |V(H)|/2$ edges, by Theorem 1, it contains a cycle of length at least $d + 1$. Let us note that the degeneracy of a graph can be computed in polynomial time, see e.g. [28], and thus by Theorem 1, deciding whether a graph has a cycle of length at least $d + 1$ can be done in polynomial time. In this paper we revisit this classical result from the algorithmic perspective.

We define the following problem.

LONGEST CYCLE ABOVE DEGENERACY

Input: A graph G and a positive integer k .

Task: Decide whether G contains a cycle of length at least $\text{dg}(G) + k$.

Let us first sketch why LONGEST CYCLE ABOVE DEGENERACY is NP-complete for $k = 2$ even for connected graphs. We can reduce HAMILTONIAN CYCLE to LONGEST CYCLE ABOVE DEGENERACY with $k = 2$ as follows. For a connected non-complete graph G on n vertices, we construct connected graph H from G and a complete graph K_{n-1} on $n - 1$ vertices as follows. We identify one vertex of G with one vertex of K_{n-1} . Thus the obtained graph H has $|V(G)| + n - 2$ vertices and is connected; its degeneracy is $n - 2$. Then H has a cycle with $\text{dg}(H) + 2 = n$ vertices if and only if G has a Hamiltonian cycle.

Interestingly, when the input graph is 2-connected, the problem becomes fixed-parameter tractable being parameterized by k . Let us recall that a connected graph G is (vertex) 2-connected if for every $v \in V(G)$, $G - v$ is connected. Our first main result is the following theorem.

► **Theorem 2.** *On 2-connected graphs LONGEST CYCLE ABOVE DEGENERACY is solvable in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.*

Similar results can be obtained for paths. Of course, if a graph contains a cycle of length $d + 1$, it also contains a simple path on $d + 1$ vertices. Thus for every graph G of degeneracy d , deciding whether G contains a path on $\text{dg}(G) + 1$ vertices can be done in polynomial time. Again, it is easy to show that it is NP-complete to decide whether G contains a path with $d + 2$ vertices by a reduction from HAMILTONIAN PATH. The reduction is very similar to the one we sketched for LONGEST CYCLE ABOVE DEGENERACY. The only difference that this time graph H consists of a disjoint union of G and K_{n-1} . The degeneracy of H is $d = n - 2$, and H has a path with $d + 2 = n$ vertices if and only if G contains a Hamiltonian path. Note that graph H used in the reduction is not connected. However, when the input graph G is connected, the complexity of the problem changes drastically. We define

LONGEST PATH ABOVE DEGENERACY

Input: A graph G and a positive integer k .
Task: Decide whether G contains a path with at least $\text{dg}(G) + k$ vertices.

The second main contribution of our paper is the following theorem.

► **Theorem 3.** *On connected graphs* LONGEST PATH ABOVE DEGENERACY *is solvable in time* $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.

Let us remark that Theorem 2 does not imply Theorem 3, because Theorem 2 holds only for 2-connected graphs.

We also show that the parameterization lower bound $\text{dg}(G)$ that is used in Theorems 2 and 3 is tight in some sense. We prove that for any $0 < \varepsilon < 1$, it is NP-complete to decide whether a connected graph G contains a path with at least $(1 + \varepsilon)\text{dg}(G)$ vertices and it is NP-complete to decide whether a 2-connected graph G contains a cycle with at least $(1 + \varepsilon)\text{dg}(G)$ vertices.

Related work. HAMILTONIAN PATH and HAMILTONIAN CYCLE problems are among the oldest and most fundamental problems in Graph Theory. In parameterized complexity the following generalizations of these problems, LONGEST PATH and LONGEST CYCLE, were heavily studied. The LONGEST PATH problem is to decide, given an n -vertex (di)graph G and an integer k , whether G contains a path of length at least k . Similarly, the LONGEST CYCLE problem is to decide whether G contains a cycle of length at least k . There is a plethora of results about parameterized complexity (we refer to the book of Cygan et al. [9] for the introduction to the field) of LONGEST PATH and LONGEST CYCLE (see, e.g., [4, 5, 7, 6, 12, 14, 22, 23, 24, 32]) since the early work of Monien [29]. The fastest known randomized algorithm for LONGEST PATH on undirected graph is due to Björklund et al. [4] and runs in time $1.657^k \cdot n^{\mathcal{O}(1)}$. On the other hand very recently, Tsur gave the fastest known deterministic algorithm for the problem running in time $2.554^k \cdot n^{\mathcal{O}(1)}$ [31]. Respectively for LONGEST CYCLE, the current fastest randomized algorithm running in time $4^k \cdot n^{\mathcal{O}(1)}$ was given by Zehavi in [33] and the best deterministic algorithm constructed by Fomin et al. in [13] runs in time $4.884^k \cdot n^{\mathcal{O}(1)}$.

Our theorems about LONGEST PATH ABOVE DEGENERACY and LONGEST CYCLE ABOVE DEGENERACY fit into an interesting trend in parameterized complexity called “above guarantee” parameterization. The general idea of this paradigm is that the natural parameterization of, say, a maximization problem by the solution size is not satisfactory if there is a lower bound for the solution size that is sufficiently large. For example, there always exists a satisfying assignment that satisfies half of the clauses or there is always a max-cut containing at least half the edges. Thus nontrivial solutions occur only for the values of the parameter that are above the lower bound. This indicates that for such cases, it is more natural to parameterize the problem by the difference of the solution size and the bound. The first paper about above guarantee parameterization was due to Mahajan and Raman [26] who applied this approach to the MAX SAT and MAX CUT problem. This approach was successfully applied to various problems, see e.g. [1, 8, 16, 17, 18, 19, 20, 25, 27].

For LONGEST PATH, the only successful above guarantee parameterization known prior to our work was parameterization above shortest path. More precisely, let s, t be vertices of an undirected graph G . Clearly, the length of any (s, t) -path in G is lower bounded by the shortest distance, $d(s, t)$, between these vertices. Based on this observation, Bezáková et al. in [3] introduced the LONGEST DETOUR problem that asks, given a graph G , two vertices

s, t , and a positive integer k , whether G has an (s, t) -path with at least $d(s, t) + k$ vertices. They proved that for undirected graphs, this problem can be solved in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$. On the other hand, the parameterized complexity of LONGEST DETOUR on directed graphs is still open. For the variant of the problem where the question is whether G has an (s, t) -path with *exactly* $d(s, t) + k$ vertices, a randomized algorithm with running time $2.746^k \cdot n^{\mathcal{O}(1)}$ and a deterministic algorithm with running time $6.745^k \cdot n^{\mathcal{O}(1)}$ were obtained [3]. These algorithms work for both undirected and directed graphs. Parameterization above degeneracy is “orthogonal” to the parameterization above the shortest distance. There are classes of graphs, like planar graphs, that have constant degeneracy and arbitrarily large diameter. On the other hand, there are classes of graphs, like complete graphs, of constant diameter and unbounded degeneracy.

Our approach. Our algorithmic results are based on classical theorems of Dirac [10], and Erdős and Gallai [11] on the existence of “long cycle” and “long paths” and can be seen as non-trivial algorithmic extensions of these classical theorems. Let $\delta(G)$ be the minimum vertex degree of graph G .

► **Theorem 4** (Dirac [10]). *Every n -vertex 2-connected graph G with minimum vertex degree $\delta(G) \geq 2$, contains a cycle with at least $\min\{2\delta(G), n\}$ vertices.*

► **Theorem 5** (Erdős and Gallai [11]). *Every connected n -vertex graph G contains a path with at least $\min\{2\delta(G) + 1, n\}$ vertices.*

Theorem 4 is used to prove Theorem 2 and Theorem 5 is used to prove Theorem 3.

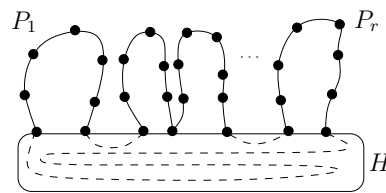
We give a high-level overview of the ideas used to prove Theorem 2. The ideas behind the proof of Theorem 3 are similar. Let G be a 2-connected graph of degeneracy d . If $d = \mathcal{O}(k)$, we can solve LONGEST CYCLE ABOVE DEGENERACY in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ by making use of one of the algorithms for LONGEST CYCLE. Assume from now that $d \geq c \cdot k$ for some constant c , which will be specified in the proof. Then we find a d -core H of G (a connected subgraph of G with the minimum vertex degree at least d). This can be done in linear time by one of the known algorithms, see e.g. [28]. If the size of H is sufficiently large, say $|V(H)| \geq d + k$, we use Theorem 4 to conclude that H contains a cycle with at least $|V(H)| \geq d + k$ vertices.

The most interesting case occurs when $|V(H)| < d + k$. Suppose that G has a cycle of length at least $d + k$. It is possible to prove that there is also a cycle of length at least $d + k$ that hits the core H . We do not know how many times and in which vertices of H this cycle enters and leaves H , but we can guess these terminal points. The interesting property of the core H is that, loosely speaking, for any “small” set of terminal points, inside H the cycle can be rerouted in such a way that it will contain all vertices of H .

A bit more formally, we prove the following structural result. We define a system of segments in G with respect to $V(H)$, which is a family of internally vertex-disjoint paths $\{P_1, \dots, P_r\}$ in G (see Figure 1). Moreover, for every $1 \leq i \leq r$, every path P_i has at least 3 vertices, its endpoints are in $V(H)$ and all internal vertices of P_i are in $V(G) \setminus V(H)$. Also the union of all the segments is a forest with every connected component being a path.

We prove that G contains a cycle of length at least $k + d$ if and only if

- either there is a path of length at least $k + d - |V(H)|$ with endpoints in $V(H)$ and all internal vertices outside H , or
- there is a system of segments with respect to $V(H)$ such that the total number of vertices outside H used by the paths of the system, is within the interval $[k + d - |V(H)|, 2 \cdot (k + d - |V(H)|)]$.



■ **Figure 1** Reducing LONGEST CYCLE ABOVE DEGENERACY to finding a system of segments P_1, \dots, P_r ; complementing the segments into a cycle is shown by dashed lines.

The proof of this structural result is built on Lemma 8, which describes the possibility of routing in graphs of large minimal degree. The crucial property is that we can complement any system of segments of bounded size by segments inside the core H to obtain a cycle that contains all the vertices of H as is shown in Figure 1.

Since $|V(H)| > d$, the problem of finding a cycle of length at least $k + d$ in G boils down to one of the following tasks. Either find a path of length $c' \cdot k$ with all internal vertices outside H , or find a system of segments with respect to $V(H)$ such that the total number of vertices used by the paths of the system is $c'' \cdot k$, here c' and c'' are the constants to be specified in the proof. In the first case, we can use one of the known algorithms to find in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ such a long path. In the second case, we can use color-coding to solve the problem.

Organization of the paper. In Section 2 we give basic definitions and state some known fundamental results. Sections 3–4 contain the proof of Theorems 3 and 2. In Section 3 we state structural results that we need for the proofs and in Section 4 we complete the proofs. In Section 5, we give the complexity lower bounds for our algorithmic results. We conclude the paper in Section 6 by stating some open problems.

2 Preliminaries

We consider only finite undirected graphs. For a graph G , we use $V(G)$ and $E(G)$ to denote its vertex set and edge set, respectively. Throughout the paper we use $n = |V(G)|$ and $m = |E(G)|$. For a graph G and a subset $U \subseteq V(G)$ of vertices, we write $G[U]$ to denote the subgraph of G induced by U . We write $G - U$ to denote the graph $G[V(G) \setminus U]$; for a single-element set $U = \{u\}$, we write $G - u$. For a vertex v , we denote by $N_G(v)$ the (*open*) *neighborhood* of v , i.e., the set of vertices that are adjacent to v in G . For a set $U \subseteq V(G)$, $N_G(U) = (\bigcup_{v \in U} N_G(v)) \setminus U$. The *degree* of a vertex v is $d_G(v) = |N_G(v)|$. The *minimum degree* of G is $\delta(G) = \min\{d_G(v) \mid v \in V(G)\}$. A d -*core* of G is an inclusion maximal induced connected subgraph H with $\delta(H) \geq d$. Every graph of degeneracy at least d contains a d -core and that can be found in linear time (see [28]). A vertex u of a connected graph G with at least two vertices is a *cut vertex* if $G - u$ is disconnected. A connected graph G is *2-connected* if it has no cut vertices. An inclusion maximal induced 2-connected subgraph of G is called a *biconnected component* or *block*. Let \mathcal{B} be the set of blocks of a connected graph G and let C be the set of cut vertices. Consider the bipartite graph $Block(G)$ with the vertex set $\mathcal{B} \cup C$, where (\mathcal{B}, C) is the bipartition, such that $B \in \mathcal{B}$ and $c \in C$ are adjacent if and only if $c \in V(B)$. The block graph of a connected graph is always a tree (see [21]).

A path in a graph is a self-avoiding walk. Thus no vertex appears in a path more than once. A cycle is a closed self-avoiding walk. For a path P with end-vertices s and t , we say that the vertices of $V(P) \setminus \{s, t\}$ are *internal*. We say that G is a *linear forest* if each component of G is a path. The *contraction* of an edge xy is the operation that removes the

vertices x and y together with the incident edges and replaces them by a vertex u_{xy} that is adjacent to the vertices of $N_G(\{x, y\})$ of the original graph. If H is obtained from G by contracting some edges, then H is a *contraction* of G .

We summarize below some known algorithmic results which will be used as subroutines by our algorithm.

► **Proposition 6.** LONGEST PATH and LONGEST CYCLE are solvable in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.

We also need the result about the variant of LONGEST PATH with fixed end-vertices. In the (s, t) -LONGEST PATH, we are given two vertices s and t of a graph G and a positive integer k . The task is to decide, whether G has an (s, t) -path with at least k vertices. Using the results of Bezáková et al. [2], we immediately obtain the following.

► **Proposition 7.** (s, t) -LONGEST PATH is solvable in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.

3 Segments and rerouting

In this section we define systems of segments and prove structural results about them. These combinatorial results are crucial for our algorithms for LONGEST PATH ABOVE DEGENERACY and LONGEST CYCLE ABOVE DEGENERACY.

The following rerouting lemma is crucial for our algorithms.

► **Lemma 8.** Let G be an n -vertex graph and k be a positive integer such that $\delta(G) \geq \max\{5k - 3, n - k\}$. Let $\{s_1, t_1\}, \dots, \{s_r, t_r\}$, $r \leq k$, be a collection of pairs of vertices of G such that (i) $s_i, t_i \notin \{s_j, t_j\}$ for all $i \neq j$, $i, j \in \{1, \dots, r\}$, and (ii) there is at least one index $i \in \{1, \dots, r\}$ such that $s_i \neq t_i$. Then there is a family of pairwise vertex-disjoint paths $\mathcal{P} = \{P_1, \dots, P_r\}$ in G such that each P_i is an (s_i, t_i) -path and $\bigcup_{i=1}^r V(P_i) = V(G)$, that is, the paths cover all vertices of G .

Proof. We prove the lemma in two steps. First we show that there exists a family \mathcal{P}' of pairwise vertex-disjoint paths connecting all pairs $\{s_i, t_i\}$. Then we show that if the paths of \mathcal{P}' do not cover all vertices of G , it is possible to enlarge a path such that the new family of paths covers more vertices.

We start by constructing a family of vertex-disjoint paths $\mathcal{P}' = \{P_1, \dots, P_r\}$ in G such that each $P_i \in \mathcal{P}'$ is an (s_i, t_i) -path. We prove that we can construct paths in such a way that each P_i has at most 3 vertices. Let $T = \bigcup_{i=1}^r \{s_i, t_i\}$ and $S = V(G) \setminus T$. Notice that $|S| \geq n - 2k \geq \delta(G) + 1 - 2k \geq 3k - 2$. We consecutively construct paths of \mathcal{P}' for $i \in \{1, \dots, r\}$. If $s_i = t_i$, then we have a trivial (s_i, t_i) -path. If s_i and t_i are adjacent, then edge $s_i t_i$ forms an (s_i, t_i) -path with 2 vertices. Assume that $s_i \neq t_i$ and $s_i t_i \notin E(G)$. The already constructed paths contain at most $r - 1 \leq k - 1$ vertices of S in total. Hence, there is a set $S' \subseteq S$ of at least $2k - 1$ vertices that are not contained in any of already constructed paths. Since $\delta(G) \geq n - k$, each vertex of G has at most $k - 1$ non-neighbors in G . By the pigeonhole principle, there is $v \in S'$ such that $s_i v, t_i v \in E(G)$. Then we can construct the path $P_i = s_i v t_i$.

We proved that there is a family $\mathcal{P}' = \{P_1, \dots, P_r\}$ of vertex-disjoint (s_i, t_i) -paths in G . Among all such families, let us select a family $\mathcal{P} = \{P_1, \dots, P_r\}$ covering the maximum number of vertices of $V(G)$. If $\bigcup_{i=1}^r V(P_i) = V(G)$, then the lemma holds. Assume that $|\bigcup_{i=1}^r V(P_i)| < |V(G)|$. Suppose $|\bigcup_{i=1}^r V(P_i)| \leq 3k - 1$. Since $s_i \neq t_i$ for some i , there is an edge uv in one of the paths. Since $n \geq \delta(G) + 1 \geq 5k - 2$, there are at least $2k - 1$ vertices uncovered by paths of \mathcal{P} . Since $\delta(G) \geq n - k$, each vertex of G has at most $k - 1$

non-neighbors in G . Thus there is $w \in V(G) \setminus (\bigcup_{i=1}^r V(P_i))$ adjacent to both u and v . But then we can extend the path containing uv by replacing uv by the path uvw . The paths of the new family cover more vertices than the paths of \mathcal{P} , which contradicts the choice of \mathcal{P} .

Suppose $|\bigcup_{i=1}^r V(P_i)| \geq 3k$. Because the paths of \mathcal{P} are vertex-disjoint, the union of edges of paths from \mathcal{P} contains a k -matching. That is, there are k edges u_1v_1, \dots, u_kv_k of G such that for every $i \in \{1, \dots, k\}$, vertices u_i, v_i are consecutive in some path from \mathcal{P} and $u_i \neq u_j, u_i \neq v_j$ for all non-equal $i, j \in \{1, \dots, k\}$. Let $w \in V(G) \setminus (\bigcup_{i=1}^r V(P_i))$. We again use the observation that w has at most $k - 1$ non-neighbors in G and, therefore, there is $j \in \{1, \dots, k\}$ such that $u_jw, v_jw \in E(G)$. Then we extend the path containing u_jv_j by replacing edge u_jv_j by the path u_jwv_j , contradicting the choice of \mathcal{P} . We conclude that the paths of \mathcal{P} cover all vertices of G . ◀

Let G be a graph and let $T \subset V(G)$ be a set of terminals. We need the following definitions.

▶ **Definition 9** (Terminal segments). *We say that a path P in G is a one-terminal T -segment if it has at least two vertices, exactly one end-vertex of P is in T and other vertices are not in T . Respectively, P is a two-terminal T -segment if it has at least three vertices, both end-vertices of P are in T and internal vertices of P are not in T .*

For every cycle C hitting H , removing the vertices of H from C turns it into a set of two-terminal T -segments for $T = V(H)$. So here is the definition.

▶ **Definition 10** (System of T -segments). *We say that a set $\{P_1, \dots, P_r\}$ of paths in G is a system of T -segments if it satisfies the following conditions.*

- (i) *For each $i \in \{1, \dots, r\}$, P_i is a two-terminal T -segment,*
- (ii) *P_1, \dots, P_r are pairwise internally vertex-disjoint, and*
- (iii) *the union of P_1, \dots, P_r is a linear forest.*

Let us remark that we do not require that the end-vertices of the paths $\{P_1, \dots, P_r\}$ cover all vertices of T . System of segments will be used for solving LONGEST CYCLE ABOVE DEGENERACY.

For LONGEST PATH ABOVE DEGENERACY we need to modify the definition of a system of T -segments to include the possibility that path can start or end in H .

▶ **Definition 11** (Extended system of T -segments). *We say that a set $\{P_1, \dots, P_r\}$ of paths in G is an extended system of T -segments if the following holds.*

- (i) *At least one and at most two paths are one-terminal T -segments and the others are two-terminal T -segments.*
- (ii) *P_1, \dots, P_r are pairwise internally vertex-disjoint and the end-vertices of each one-terminal segment that is in $V(G) \setminus T$ is pairwise distinct with the other vertices of the paths.*
- (iii) *The union of P_1, \dots, P_r is a linear forest and if $\{P_1, \dots, P_r\}$ contains two one-terminal segments, then the vertices of these segments are in distinct components of the forest.*

The following lemma will be extremely useful for the algorithm solving LONGEST PATH ABOVE DEGENERACY. Informally, it shows that if a connected graph G is of large degeneracy but has a small core H , then deciding whether G has a path of length $k + d$ can be reduced to checking whether G has an extended system of T -segments with terminal set $T = V(H)$ such that the total number of vertices used by the system is $\mathcal{O}(k)$.

► **Lemma 12.** *Let $d, k \in \mathbb{N}$. Let G be a connected graph with a d -core H such that $d \geq 5k - 3$ and $d > |V(H)| - k$. Then G has a path on $d + k$ vertices if and only if G has an extended system of T -segments $\{P_1, \dots, P_r\}$ with terminal set $T = V(H)$ such that the total number of vertices contained in the paths of the system in $V(G) \setminus V(H)$ is $p = d + k - |V(H)|$.*

Proof. We put $T = V(H)$. Suppose first that G has an extended system $\{P_1, \dots, P_r\}$ of T -segments and that the total number of vertices of the paths in the system outside T is $p = d + k - |T|$. Let s_i and t_i be the end-vertices of P_i for $i \in \{1, \dots, r\}$ and assume without loss of generality that for $1 \leq i < j \leq r$, the vertices of P_i and P_j are pairwise distinct with the possible exception $t_i = s_j$ when $i = j - 1$. We also assume without loss of generality that P_1 is a one-terminal segment and $t_1 \in T$ and if $\{P_1, \dots, P_r\}$ has two one-terminal segments, then the second such segment is P_r and $s_r \in T$. Note that because $|V(H)| > d$, we have that $p = d + k - |V(H)| < k$.

Suppose that $\{P_1, \dots, P_r\}$ contains one one-terminal segment P_1 . Let s_{r+1} be an arbitrary vertex of $T \setminus (\bigcup_{i=1}^r V(P_i))$. Notice that such a vertex exists, because $|T \cap (\bigcup_{i=1}^r V(P_i))| \leq 2p - 1 < 2k - 1$ and $|T| \geq d + 1 \geq 5k - 3$. Consider the collection of pairs of vertices $\{t_1, s_2\}, \{t_2, s_3\}, \dots, \{t_r, s_{r+1}\}$. Notice that vertices from distinct pairs are distinct and $t_r \neq s_{r+1}$. By Lemma 8, there are vertex-disjoint paths P'_1, \dots, P'_r in H that cover T such that P'_i is a (t_i, s_{i+1}) -path for $i \in \{1, \dots, r\}$. By concatenating $P_1, P'_1, P_2, \dots, P_r, P'_r$ we obtain a path in G with $|T| + p = d + k$ vertices.

Assume now that $\{P_1, \dots, P_r\}$ contains two one-terminal segments P_1 and P_r . Consider the collection of pairs of vertices $\{t_1, s_2\}, \dots, \{t_{r-1}, s_r\}$. Notice that vertices from distinct pairs are distinct and there is $i \in \{2, \dots, r\}$ such that $t_{i-1} \neq s_i$ by the condition (iii) of the definition of an extended system of segments. By Lemma 8, there are vertex-disjoint paths P'_1, \dots, P'_{r-1} in H that cover T such that P'_i is a (t_i, s_{i+1}) -path for $i \in \{1, \dots, r - 1\}$. By concatenating $P_1, P'_1, \dots, P'_{r-1}, P_r$ we obtain a path in G with $|T| + p = d + k$ vertices.

To show the implication in the opposite direction, let us assume that G has an (x, y) -path P with $d + k$ vertices. We distinguish several cases.

Case 1: $V(P) \cap T = \emptyset$. Consider a shortest path P' with one end-vertex $s \in V(P)$ and the second end-vertex $t \in T$. Notice that such a path exists, because G is connected. Denote by P_x and P_y the (s, x) and (s, y) -subpaths of P respectively. Because $d \geq 5k - 3$, $|V(P_x)| \geq k$ or $|V(P_y)| \geq k$. Assume that $|V(P_x)| \geq k$. Then the concatenation of P' and P_x is a path with at least $k + 1$ vertices and it contains a subpath P'' with the end-vertex t with $p + 1$ vertices. We have that $\{P'\}$ is an extended system of T -segments and P'' has p vertices outside T .

Case 2: $V(P) \cap T \neq \emptyset$ and $E(P) \cap E(H) = \emptyset$. Let $S = V(P) \cap T$. Since H is an induced subgraph of G and $E(P) \cap E(H) = \emptyset$, $|V(P) \setminus S| \geq (d + k)/2 - 1 \geq 3k - 5/2 > 3p - 5/2 \geq 2p - 2$. Then for every $t \in S$, either the (t, x) -subpath P_x of P contains at least p vertices outside T or the (t, y) -subpath P_y of P contains at least p vertices outside T . Assume without loss of generality that P_x contains at least p vertices outside T . Consider the minimal subpath P' of P_x ending at t such that $|V(P') \setminus T| = p$. Then the start vertex s of P' is not in T . Let $\{t_1, \dots, t_r\} = V(P') \cap T$ and assume that t_1, \dots, t_r are ordered in the same order as they occur in P' starting from s . In particular, $t_r = t$. Let $t_0 = s$. Consider the paths P_1, \dots, P_r where P_i is the (t_{i-1}, t_i) -subpath of P' for $i \in \{1, \dots, r\}$. Since $k > p$, $r \leq k$. We obtain that $\{P_1, \dots, P_r\}$ is an extended system of T -segments with p vertices outside T .

Case 3: $E(P) \cap E(H) \neq \emptyset$. Then there are distinct $s, t \in T \cap V(P)$ such that the (s, t) -subpath of P lies in H . Since P has at least p vertices outside T , there are $s', t' \in V(P) \setminus T$ such that the (s', t') -subpath P' of P is a subpath with exactly p vertices outside T with $s, t \in V(P')$. Let P_1, \dots, P_r be the family of inclusion maximal subpaths of P' containing

the vertices of $V(P') \setminus T$ such that the internal vertices of each P_i are outside T . Observe that since $s \neq t$, the union of these paths is a linear forest with at least two components. We conclude that the set $\{P_1, \dots, P_r\}$ is a required extended system of T -segments. ◀

The next lemma will be used for solving LONGEST CYCLE ABOVE DEGENERACY.

► **Lemma 13.** *Let $d, k \in \mathbb{N}$. Let G be a 2-connected graph with a d -core H such that $d \geq 5k - 3$ and $d > |V(H)| - k$. Then G has a cycle with at least $d + k$ vertices if and only if one of the following holds (where $p = d + k - |V(H)|$).*

- (i) *There are distinct $s, t \in V(H)$ and an (s, t) -path P in G with all internal vertices outside $V(H)$ such that P has at least p internal vertices.*
- (ii) *G has a system of T -segments $\{P_1, \dots, P_r\}$ with terminal set $T = V(H)$ and the total number of vertices of the paths outside $V(H)$ is at least p and at most $2p - 2$.*

Proof. We put $T = V(H)$. First, we show that if (i) or (ii) holds, then G has a cycle with at least $d + k$ vertices. Suppose that there are distinct $s, t \in T$ and an (s, t) -path P in G with all internal vertices outside T such that P has at least p internal vertices. By Lemma 8, H has a Hamiltonian (s, t) -path P' . By taking the union of P and P' we obtain a cycle with at least $|T| + p = d + k$ vertices.

Now assume that G has a system of T -segments $\{P_1, \dots, P_r\}$ and the total number of vertices of the paths outside T is at least p . Let s_i and t_i be the end-vertices of P_i for $i \in \{1, \dots, r\}$ and assume without loss of generality that for $1 \leq i < j \leq r$, the vertices of P_i and P_j are pairwise distinct with the possible exception $t_i = s_j$ when $i = j - 1$. Consider the collection of pairs of vertices $\{t_1, s_2\}, \dots, \{t_{r-1}, s_r\}, \{t_r, s_1\}$. Notice that vertices from distinct pairs are distinct and $t_r \neq s_1$. By Lemma 8, there are vertex-disjoint paths P'_1, \dots, P'_r in H that cover T such that P'_i is a (t_i, s_{i+1}) -path for $i \in \{1, \dots, r - 1\}$ and P'_r is a (t_r, s_1) -path. By taking the union of P_1, \dots, P_r and P'_1, \dots, P'_r we obtain a cycle in G with at least $|T| + p = d + k$ vertices.

To show the implication in the other direction, assume that G has a cycle C with at least $d + k$ vertices.

Case 1: $V(C) \cap T = \emptyset$. Since G is a 2-connected graph, there are pairwise distinct vertices $s, t \in T$ and $x, y \in V(C)$ and vertex-disjoint (s, x) and (y, t) -paths P_1 and P_2 such that the internal vertices of the paths are outside $T \cup V(C)$. The cycle C contains an (x, y) -path P with at least $(d + k)/2 + 1 \geq p$ vertices. The concatenation of P_1, P and P_2 is an (s, t) -path in G with at least p internal vertices and the internal vertices are outside T . Hence, (i) holds.

Case 2: $|V(C) \cap T| = 1$. Let $V(C) \cap T = \{s\}$ for some vertex s . Since G is 2-connected, there is a shortest (x, t) -path P in $G - s$ such that $x \in V(C)$ and $t \in T$. The cycle C contains an (s, x) -path P' with at least $(d + k)/2 + 1 \geq p$ vertices. The concatenation of P' and P is an (s, t) -path in G with at least p internal vertices and the internal vertices of the path are outside T . Therefore, (i) is fulfilled.

Case 3: $|V(C) \cap T| \geq 2$. Since $|V(C)| \geq d$ and $|T| < d$, we have that $V(C) \setminus T \neq \emptyset$. Then we can find pairs of distinct vertices $\{s_1, t_1\}, \dots, \{s_\ell, t_\ell\}$ of $T \cap V(C)$ and segments P_1, \dots, P_ℓ of C such that (a) P_i is an (s_i, t_i) -path for $i \in \{1, \dots, \ell\}$ with at least one internal vertex and the internal vertices of P_i are outside T , (b) for $1 \leq i < j \leq \ell$, the vertices of P_i and P_j are distinct with the possible exception $t_i = s_j$ if $i = j - 1$ and, possibly, $t_\ell = s_1$, and (c) $\bigcup_{i=1}^{\ell} V(P_i) \setminus T = V(C) \setminus T$. If there is $i \in \{1, \dots, \ell\}$ such that P_i has at least p internal vertices, then (i) is fulfilled.

47:10 Going Far From Degeneracy

Now assume that each P_i has at most $p - 1$ internal vertices; notice that $p \geq 2$ in this case. We select an inclusion minimal set of indices $I \subseteq \{1, \dots, \ell\}$ such that $|\bigcup_{i \in I} V(P_i) \setminus T| \geq p$. Notice that because each path has at most $p - 1$ internal vertices, $|\bigcup_{i \in I} V(P_i) \setminus T| \leq 2p - 2$. Let $I = \{i_1, \dots, i_r\}$ and $i_1 < \dots < i_r$. By the choice of P_{i_1}, \dots, P_{i_r} , the union of P_{i_1}, \dots, P_{i_r} is either the cycle C or a linear forest. Suppose that the union of the paths is C . Then $I = \{1, \dots, \ell\}$, $\ell \leq p$ and $|V(P) \cap T| = \ell$. Note that because $|V(H)| > d$, we have that $p = d + k - |V(H)| < k$. We obtain that C has at most $(2p - 2) + p \leq 3p - 2 < 3k - 2 < d + k$ vertices (the last inequality follows from the fact that $d \geq 5k - 3$); a contradiction. Hence, the union of the paths is a linear forest. Therefore, $\{P_{i_1}, \dots, P_{i_r}\}$ is a system of T -segments with terminal set $T = V(H)$ and the total number of vertices of the paths outside T is at least p and at most $2p - 2$, that is, (ii) is fulfilled. ◀

We have established the fact that the existence of a long (path) cycle is equivalent to the existence of an (extended) system of T -segments for some terminal set T with at most $p \leq k$ vertices from outside T . Towards designing algorithms for LONGEST PATH ABOVE DEGENERACY and LONGEST CYCLE ABOVE DEGENERACY, we define two auxiliary problems which can be solved using well known color-coding technique.

SEGMENTS WITH TERMINAL SET

Input: A graph G , $T \subset V(G)$ and a positive integers p and r .
Task: Decide whether G has a system of segments $\{P_1, \dots, P_r\}$ w.r.t. T such that the total number of internal vertices of the paths is p .

EXTENDED SEGMENTS WITH TERMINAL SET

Input: A graph G , $T \subset V(G)$ and a positive integers p and r .
Task: Decide whether G has an extended system of segments $\{P_1, \dots, P_r\}$ w.r.t. T such that the total number of vertices of the paths outside T is p .

► **Lemma 14** (*).¹ SEGMENTS WITH TERMINAL SET and EXTENDED SEGMENTS WITH TERMINAL SET are solvable in time $2^{\mathcal{O}(p)} \cdot n^{\mathcal{O}(1)}$.

4 Putting all together: Final proofs

Proof of Theorem 3. Let G be a connected graph of degeneracy at least d and let k be a positive integer. If $d \leq 5k - 4$, then we check the existence of a path with $d + k \leq 6k - 4$ vertices using Proposition 6 in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$. Assume from now that $d \geq 5k - 3$. Then we find a d -core H of G . This can be done in linear time using the results of Matula and Beck [28]. If $|V(H)| \geq d + k$, then by Theorem 5, H , and hence G , contains a path with $\min\{2d + 1, |V(H)|\} \geq d + k$ vertices. Assume that $|V(H)| < d + k$. By Lemma 12, G has a path with $d + k$ vertices if and only if G has paths P_1, \dots, P_r such that $\{P_1, \dots, P_r\}$ is an extended system of T -segments for $T = V(H)$ and the total number of vertices of the paths outside T is $p = d + k - |T|$. Since the number of vertices in every graph is more than its minimum degree, we have that $|T| > d$, and thus $p < k$. For each $r \in \{1, \dots, p\}$, we verify if such a system exists in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ by making use of Lemma 14. Thus the total running time of the algorithm is $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.

¹ Proofs of results marked with (*) are omitted in this extended abstract.

Proof of Theorem 2. Let G be a 2-connected graph of degeneracy at least d and let $k \in \mathbb{N}$. If $d \leq 5k - 4$, then we check the existence of a cycle with at least $d + k \leq 6k - 4$ vertices using Proposition 6 in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$. Assume from now on that $d \geq 5k - 3$. Then we find a d -core H of G in linear time using the results of Matula and Beck [28].

We claim that if $|V(H)| \geq d + k$, then H contains a cycle with at least $d + k$ vertices. If H is 2-connected, then this follows from Theorem 4. Assume that H is not a 2-connected graph. By the definition of a d -core, H is connected. Observe that $|V(H)| \geq d + 1 \geq 5k - 2 \geq 3$. Hence, H has at least two blocks and at least one cut vertex. Consider the block graph $Block(H)$ of H . Recall that the vertices of $Block(H)$ are the blocks and the cut vertices of H and a cut vertex c is adjacent to a block B if and only if $c \in V(B)$. Recall also that $Block(H)$ is a tree. We select an arbitrary block R of H and declare it to be the *root* of $Block(H)$. Let $S = V(G) \setminus V(H)$. Observe that $S \neq \emptyset$, because G is 2-connected and H is not. Let F_1, \dots, F_ℓ be the components of $G[S]$. We contract the edges of each component and denote the obtained vertices by u_1, \dots, u_ℓ . Denote by G' the obtained graph. It is straightforward to verify that G' has no cut vertices, that is, G' is 2-connected. For each $i \in \{1, \dots, \ell\}$, consider u_i . This vertex has at least 2 neighbors in $V(H)$. We select a vertex $v_i \in N_{G'}(u_i)$ that is not a cut vertex of H or, if all the neighbors of u_i are cut vertices, we select v_i be a cut vertex at maximum distance from R in $Block(H)$. Then we contract $u_i v_i$. Observe that by the choice of each v_i , the graph G'' obtained from G' by contracting $u_1 v_1, \dots, u_\ell v_\ell$ is 2-connected. We have that G'' is a 2-connected graph of minimum degree at least d with at least $d + k$ vertices. By Theorem 4, G'' has a cycle with at least $\min\{2d, |V(G'')|\} \geq d + k$ vertices. Because G'' is a contraction of G , we conclude that G contains a cycle with at least $d + k$ vertices as well.

From now we can assume that $|V(H)| < d + k$. By Lemma 13, G has a cycle with $d + k$ vertices if and only if one of the following holds for $p = d + k - |T|$ where $T = V(H)$.

- (i) There are distinct $s, t \in T$ and an (s, t) -path P in G with all internal vertices outside T such that P has at least p internal vertices.
- (ii) G has a system of T -segments $\{P_1, \dots, P_r\}$ and the total number of vertices of the paths outside T is at least p and at most $2p - 2$.

Notice that $p \leq k$ (because $d - |T| \leq 0$). We verify whether (i) holds using Proposition 7. To do it, we consider all possible choices of distinct s, t . Then we construct the auxiliary graph G_{st} from G by the deletion of the vertices of $T \setminus \{s, t\}$ and the edges of $E(H)$. Then we check whether G_{st} has an (s, t) -path of length at least $p + 1$ in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ applying Proposition 7.

Assume that (i) is not fulfilled. Then it remains to check (ii). For every $r \in \{1, \dots, p\}$, we verify the existence of a system of T -segments $\{P_1, \dots, P_r\}$ in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ using Lemma 14. We return the answer *yes* if we get the answer *yes* for at least one instance of SEGMENTS WITH TERMINAL SET and we return *no* otherwise.

5 Hardness for Longest Path and Cycle above Degeneracy

In this section we complement Theorems 3 and 2 by some hardness observations.

► **Proposition 15** (\star). ² LONGEST PATH ABOVE DEGENERACY is NP-complete even if $k = 2$ and LONGEST CYCLE ABOVE DEGENERACY is NP-complete even for connected graphs and $k = 2$.

² Proposition 15 and its proof was pointed to us by Nikolay Karpov.

47:12 Going Far From Degeneracy

Recall that a graph G has a path with at least $\text{dg}(G) + 1$ vertices and if $\text{dg}(G) \geq 2$, then G has a cycle with at least $\text{dg}(G) + 1$ vertices. Moreover, such a path or cycle can be constructed in polynomial (linear) time. Hence, Proposition 15 gives tight complexity bounds. Nevertheless, the construction used to show hardness for LONGEST PATH ABOVE DEGENERACY uses a disconnected graph, and the graph constructed to show hardness for LONGEST CYCLE ABOVE DEGENERACY has a cut vertex. Hence, it is natural to consider LONGEST PATH ABOVE DEGENERACY for connected graphs and LONGEST CYCLE ABOVE DEGENERACY for 2-connected graphs. We show in Theorems 3 and 2 that these problems are FPT when parameterized by k in these cases. Here, we observe that the lower bound $\text{dg}(G)$ that is used for the parameterization is tight in the following sense.

► **Proposition 16.** *For any $0 < \varepsilon < 1$, it is NP-complete to decide whether a connected graph G contains a path with at least $(1 + \varepsilon)\text{dg}(G)$ vertices and it is NP-complete to decide whether a 2-connected graph G contains a cycle with at least $(1 + \varepsilon)\text{dg}(G)$ vertices.*

Proof. Let $0 < \varepsilon < 1$.

First, we consider the problem about a path with $(1 + \varepsilon)\text{dg}(G)$ vertices. We reduce HAMILTONIAN PATH that is well-known to be NP-complete (see [15]). Let G be a graph with $n \geq 2$ vertices. We construct the graph G' as follows.

- Construct a copy of G .
- Let $p = 2\lceil \frac{n}{\varepsilon} \rceil$ and construct p pairwise adjacent vertices u_1, \dots, u_p .
- For each $v \in V(G)$, construct an edge vu_1 .
- Let $q = \lceil (1 + \varepsilon)(p - 1) - (n + p) \rceil$. Construct vertices w_1, \dots, w_q and edges $u_1w_1, w_q u_2$ and $w_{i-1}w_i$ for $i \in \{2, \dots, q\}$.

Notice that $q = \lceil (1 + \varepsilon)(p - 1) - (n + p) \rceil = \lceil 2\varepsilon\lceil \frac{n}{\varepsilon} \rceil - n - 1 - \varepsilon \rceil \geq \lceil n - 1 - \varepsilon \rceil \geq 1$ as $n \geq 2$. Observe also that G is connected. We claim that G has a Hamiltonian path if and only if G' has a path with at least $(1 + \varepsilon)\text{dg}(G')$ vertices. Notice that $\text{dg}(G') = p - 1$ and $|V(G')| = n + p + q = \lceil (1 + \varepsilon)\text{dg}(G') \rceil$. Therefore, we have to show that G has a Hamiltonian path if and only if G' has a Hamiltonian path. Suppose that G has a Hamiltonian path P with an end-vertex v . Consider the path $Q = vu_1w_1 \dots w_q u_2u_3 \dots u_p$. Clearly, the concatenation of P and Q is a Hamiltonian path in G' . Suppose that G' has a Hamiltonian path P . Since u_1 is a cut vertex of G' , we obtain that P has a subpath that is a Hamiltonian path in G .

Consider now the problem about a cycle with at least $(1 + \varepsilon)\text{dg}(G)$ vertices. We again reduce HAMILTONIAN PATH and the reduction is almost the same. Let G be a graph with $n \geq 2$ vertices. We construct the graph G' as follows.

- Construct a copy of G .
- Let $p = 2\lceil \frac{n}{\varepsilon} \rceil$ and construct p pairwise adjacent vertices u_1, \dots, u_p .
- For each $v \in V(G)$, construct edges vu_1 and vu_2 .
- Let $q = \lceil (1 + \varepsilon)(p - 1) - (n + p) \rceil$. Construct vertices w_1, \dots, w_q and edges $u_2w_1, w_q u_3$ and $w_{i-1}w_i$ for $i \in \{2, \dots, q\}$.

As before, we have that $q \geq 1$. Notice additionally that $p \geq 3$, i.e., the vertex u_3 exists. It is straightforward to see that G' is 2-connected. We claim that G has a Hamiltonian path if and only if G' has a cycle with at least $(1 + \varepsilon)\text{dg}(G')$ vertices. We have that $\text{dg}(G') = p - 1$ and $|V(G')| = \lceil (1 + \varepsilon)\text{dg}(G') \rceil$. Hence, we have to show that G has a Hamiltonian path if and only if G' has a Hamiltonian cycle. Suppose that G has a Hamiltonian path P with end-vertices x and y . Consider the path $Q = xu_2w_1 \dots w_q u_3u_4 \dots u_p y$. Clearly, P and Q together form a Hamiltonian cycle in G' . Suppose that G' has a Hamiltonian cycle C . Since $\{u_1, u_2\}$ is a cut set of G' , we obtain that C contains a path that is a Hamiltonian path of G . ◀

6 Conclusion

We considered the lower bound $\text{dg}(G) + 1$ for the number of vertices in a longest path or cycle in a graph G . It would be interesting to consider the lower bounds given in Theorems 4 and 5. More precisely, what can be said about the parameterized complexity of the variants of LONG PATH (CYCLE) where given a (2-connected) graph G and $k \in \mathbb{N}$, the task is to check whether G has a path (cycle) with at least $2\delta(G) + k$ vertices? Are these problems FPT when parameterized by k ? It can be observed that the bound $2\delta(G)$ is “tight”. That is, for any $0 < \varepsilon < 1$, it is NP-complete to decide whether a connected (2-connected) G has a path (cycle) with at least $(2 + \varepsilon)\delta(G)$ vertices. See also [30] for related hardness results.

References

- 1 Noga Alon, Gregory Gutin, Eun Jung Kim, Stefan Szeider, and Anders Yeo. Solving MAX- r -SAT Above a Tight Lower Bound. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 511–517. SIAM, 2010.
- 2 Ivona Bezáková, Radu Curticapean, Holger Dell, and Fedor V. Fomin. Finding Detours is Fixed-parameter Tractable. *CoRR*, abs/1607.07737, 2016. URL: <http://arxiv.org/abs/1607.07737>, arXiv:1607.07737.
- 3 Ivona Bezáková, Radu Curticapean, Holger Dell, and Fedor V. Fomin. Finding Detours is Fixed-Parameter Tractable. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, volume 80 of *LIPICs*, pages 54:1–54:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.
- 4 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *CoRR*, abs/1007.1161, 2010. URL: <http://arxiv.org/abs/1007.1161>.
- 5 Hans L. Bodlaender. On Linear Time Minor Tests with Depth-First Search. *J. Algorithms*, 14(1):1–23, 1993. doi:10.1006/jagm.1993.1001.
- 6 Jianer Chen, Joachim Kneis, Songjian Lu, Daniel Mölle, Stefan Richter, Peter Rossmanith, Sing-Hoi Sze, and Fenghui Zhang. Randomized divide-and-conquer: improved path, matching, and packing algorithms. *SIAM J. Comput.*, 38(6):2526–2547, 2009. doi:10.1137/080716475.
- 7 Jianer Chen, Songjian Lu, Sing-Hoi Sze, and Fenghui Zhang. Improved algorithms for path, matching, and packing problems. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 298–307. SIAM, 2007.
- 8 Robert Crowston, Mark Jones, Gabriele Muciaccia, Geevarghese Philip, Ashutosh Rai, and Saket Saurabh. Polynomial Kernels for lambda-extendible Properties Parameterized Above the Poljak-Turzic Bound. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 24 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43–54, Dagstuhl, Germany, 2013. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
- 9 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 10 G. A. Dirac. Some theorems on abstract graphs. *Proc. London Math. Soc.* (3), 2:69–81, 1952.
- 11 P. Erdős and T. Gallai. On maximal paths and circuits of graphs. *Acta Math. Acad. Sci. Hungar.*, 10:337–356 (unbound insert), 1959.
- 12 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.
- 13 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Long directed (s, t) -path: FPT algorithm. *Inf. Process. Lett.*, 140:8–12, 2018. doi:10.1016/j.ipl.2018.04.018.

- 14 Harold N. Gabow and Shuxin Nie. Finding a long directed cycle. *ACM Transactions on Algorithms*, 4(1), 2008. doi:10.1145/1328911.1328918.
- 15 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 16 Shivam Garg and Geevarghese Philip. Raising The Bar For Vertex Cover: Fixed-parameter Tractability Above a Higher Guarantee. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1152–1166. SIAM, 2016. doi:10.1137/1.9781611974331.ch80.
- 17 Gregory Gutin, Eun Jung Kim, Michael Lampis, and Valia Mitsou. Vertex Cover Problem Parameterized Above and Below Tight Bounds. *Theory of Computing Systems*, 48(2):402–410, 2011. doi:10.1007/s00224-010-9262-y.
- 18 Gregory Gutin, Leo van Iersel, Matthias Mnich, and Anders Yeo. Every ternary permutation constraint satisfaction problem parameterized above average has a kernel with a quadratic number of variables. *J. Computer and System Sciences*, 78(1):151–163, 2012. doi:10.1016/j.jcss.2011.01.004.
- 19 Gregory Z. Gutin and Viresh Patel. Parameterized Traveling Salesman Problem: Beating the Average. *SIAM J. Discrete Math.*, 30(1):220–238, 2016.
- 20 Gregory Z. Gutin, Arash Rafiey, Stefan Szeider, and Anders Yeo. The Linear Arrangement Problem Parameterized Above Guaranteed Value. *Theory Comput. Syst.*, 41(3):521–538, 2007. doi:10.1007/s00224-007-1330-6.
- 21 Frank Harary. A characterization of block-graphs. *Canad. Math. Bull.*, 6:1–6, 1963.
- 22 Falk Hüffner, Sebastian Wernicke, and Thomas Zichner. Algorithm Engineering for Color-Coding with Applications to Signaling Pathway Detection. *Algorithmica*, 52(2):114–132, 2008. doi:10.1007/s00453-007-9008-7.
- 23 Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. Divide-and-Color. In *Proceedings of the 34th International Workshop Graph-Theoretic Concepts in Computer Science (WG)*, volume 4271 of *Lecture Notes in Computer Science*, pages 58–67. Springer, 2008.
- 24 Ioannis Koutis. Faster Algebraic Algorithms for Path and Packing Problems. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 5125 of *Lecture Notes in Comput. Sci.*, pages 575–586. Springer, 2008.
- 25 Daniel Lokshantov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster Parameterized Algorithms Using Linear Programming. *ACM Trans. Algorithms*, 11(2):15:1–15:31, 2014. doi:10.1145/2566616.
- 26 Meena Mahajan and Venkatesh Raman. Parameterizing above Guaranteed Values: MaxSat and MaxCut. *J. Algorithms*, 31(2):335–354, 1999.
- 27 Meena Mahajan, Venkatesh Raman, and Somnath Sikdar. Parameterizing above or below guaranteed values. *J. Computer and System Sciences*, 75(2):137–153, 2009. doi:10.1016/j.jcss.2008.08.004.
- 28 David W. Matula and Leland L. Beck. Smallest-Last Ordering and clustering and Graph Coloring Algorithms. *J. ACM*, 30(3):417–427, 1983. doi:10.1145/2402.322385.
- 29 B. Monien. How to find long paths efficiently. In *Analysis and design of algorithms for combinatorial problems (Udine, 1982)*, volume 109 of *North-Holland Math. Stud.*, pages 239–254. North-Holland, Amsterdam, 1985. doi:10.1016/S0304-0208(08)73110-4.
- 30 Ingo Schiermeyer. Problems remaining NP-complete for sparse or dense graphs. *Discussiones Mathematicae Graph Theory*, 15(1):33–41, 1995. doi:10.7151/dmgt.1004.
- 31 Dekel Tsur. Faster deterministic parameterized algorithm for k-Path. *CoRR*, abs/1808.04185, 2018. arXiv:1808.04185.
- 32 Ryan Williams. Finding Paths of Length k in $O^*(2^k)$ Time. *Inf. Process. Lett.*, 109(6):315–318, 2009.
- 33 Meirav Zehavi. A randomized algorithm for long directed cycle. *Inf. Process. Lett.*, 116(6):419–422, 2016. doi:10.1016/j.ipl.2016.02.005.

Group Activity Selection with Few Agent Types

Robert Ganian

TU Wien, Vienna, Austria
ganian@ac.tuwien.ac.at

Sebastian Ordyniak

University of Sheffield, Sheffield, UK
sordyniak@gmail.com

C. S. Rahul

University of Warsaw, Warsaw, Poland
rahulcsbn@gmail.com

Abstract

The Group Activity Selection Problem (GASP) models situations where a group of agents needs to be distributed to a set of activities while taking into account preferences of the agents w.r.t. individual activities and activity sizes. The problem, along with its well-known variants sGASP and gGASP, has previously been studied in the parameterized complexity setting with various parameterizations, such as *number of agents*, *number of activities* and *solution size*. However, the complexity of the problem parameterized by the *number of types of agents*, a natural parameter proposed already in the first paper that introduced GASP, has so far remained unexplored.

In this paper we establish the complexity map for GASP, sGASP and gGASP when the number of types of agents is the parameter. Our positive results, consisting of one fixed-parameter algorithm and one XP algorithm, rely on a combination of novel Subset Sum machinery (which may be of general interest) and identifying certain *compression steps* which allow us to focus on solutions which are “acyclic”. These algorithms are complemented by matching lower bounds, which among others close a gap to a recently obtained tractability result of Gupta, Roy, Saurabh and Zehavi (2017). In this direction, the techniques used to establish W[1]-hardness of sGASP are of particular interest: as an intermediate step, we use Sidon sequences to show the W[1]-hardness of a highly restricted variant of multi-dimensional Subset Sum, which may find applications in other settings as well.

2012 ACM Subject Classification Theory of computation → Exact and approximate computation of equilibria; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases group activity selection problem, parameterized complexity analysis, multi-agent systems

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.48

Related Version A full version of the paper is available at <https://arxiv.org/abs/1808.06954>.

Acknowledgements Robert Ganian acknowledges support by the Austrian Science Fund (FWF, Project P31336), and is also affiliated with FI MUNI, Czech Republic.

1 Introduction

In this paper we consider the GROUP ACTIVITY SELECTION PROBLEM (GASP) together with its two most prominent variants, the SIMPLE GROUP ACTIVITY SELECTION PROBLEM (sGASP), and the GROUP ACTIVITY SELECTION PROBLEM WITH GRAPH STRUCTURE (gGASP) [6, 18]. Since their introduction, these problems have received considerable attention, notably in venues dedicated to multi-agent systems and game theory [3, 4, 5, 19, 14, 15]. In GASP one is given a set of agents, a set of activities, and a set of preferences for each agent in the form of a complete transitive relation (also called the *preference list*) over the set of pairs consisting of an activity a and a number s , expressing the willingness of the agent to



© Robert Ganian, Sebastian Ordyniak, and C. S. Rahul;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 48; pp. 48:1–48:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

participate in the activity a if it has s participants. The aim is to find a “good” assignment from agents to activities subject to certain *rationality* and *stability* conditions. Specifically, an assignment is individually rational if agents that are assigned to an activity prefer this outcome over not being assigned to any activity, and an assignment is (Nash) stable if every agent prefers its current assignment over moving to any other activity. In this way GASP captures a wide range of real-life situations such as event organization and work delegation.

sGASP is a simplified variant of GASP where the preferences of agents are expressed in terms of *approval sets* containing (activity, size) pairs instead of preference lists. In essence sGASP is GASP where each preference list has only two equivalence classes: the class of the approved (activity, size) pairs (which contains all pairs that are preferred over not being assigned to any activity), and the class of disapproved (activity, size) pairs (all possible remaining pairs). On the other hand, gGASP is a generalization of GASP where one is additionally given an undirected graph (network) on the set of all agents that can be employed to model for instance acquaintanceship or physical distance between agents. Crucially, in gGASP one only considers assignments for which the subnetwork induced by all agents assigned to some activity is connected. Note that if the network forms a complete graph, then gGASP is equivalent to the underlying GASP instance.

Related Work. sGASP, GASP, and gGASP, are known to be NP-complete even in very restricted settings [6, 18, 14, 15]. It is therefore natural to study these problems through the lens of parameterized complexity [8, 2]. Apart from parameterizing by the solution size (i.e., the number of agents assigned to any activity in a solution) [19], the perhaps most prominent parameterizations thus far have been the number of activities, the number of agents, and in the case of gGASP parameterizations tied to the structure of the network [6, 17, 18, 14, 9]. Consequently, the parameterized complexity of all three variants of GASP w.r.t. the number of activities and/or the number of agents is now almost completely understood.

Namely, computing a stable assignment for a given instance of GASP is known to be W[1]-hard and contained in XP parameterized by either the number of activities [6, 17, 15] or the number of agents [18, 15] and known to be fixed-parameter tractable parameterized by both parameters [17, 15]. Even though it has never been explicitly stated, the same results also hold for gGASP when parameterizing by the number of agents as well as when using both parameters. This is because both the XP algorithm for the number of agents as well as the fixed-parameter algorithm for both parameters essentially brute-force over every possible assignment and are hence also able to find a solution for gGASP. Moreover, the fact that gGASP generalizes GASP implies that the W[1]-hardness result for the number of agents also carries over to gGASP.

On the other hand, if we consider the number of activities as a parameter then gGASP turns out to be harder than GASP: Gupta et al. ([14]) showed that gGASP is NP-complete even when restricted to instances with a single activity. The hardness of gGASP has inspired a series of tractability results [14, 18, 17] obtained by employing additional restrictions on the structure of the network. One prominent result in this direction has been recently obtained by Gupta et al. ([14]), showing that gGASP is fixed-parameter tractable parameterized by the number of activities if the network has constant treewidth.

Already with the introduction of GASP [6] the authors argued that instead of putting restrictions on the total number of agents, which can be very large in general, it might be much more useful to consider the number of distinct types of agents. It is easy to imagine a setting with large groups of agents that share the same preferences (for instance due to inherent limitations of how preferences are collected). In contrast to the related parameter number of activity types, where it is known that sGASP remains NP-complete even for a

constant number of activity types [6], the complexity of the problems parameterized by the number of agent types (with or without restricting the number of activities) has remained wide open thus far.

Our Results. In this paper we obtain a complete classification of the complexity of GASP and its variants sGASP and gGASP when parameterized by the number of agent types (t) alone, and also when parameterized by t plus the number of activities (a). In particular, for each of the considered problems and parameterizations, we determine whether the problem is in FPT, or W[1]-hard and in XP, or paraNP-hard. One distinguishing feature of our lower- and upper-bound results is that they make heavy use of novel Subset-Sum machinery. Below, we provide a high-level summary of the individual results presented in the paper.

Result 1. sGASP is fixed-parameter tractable when parameterized by $t + a$

This is the only fixed-parameter tractability result presented in the paper, and is essentially tight: it was recently shown that sGASP is W[1]-hard when parameterized by a alone [9], and the W[1]-hardness of the problem when parameterized by t is obtained in this paper. Our first step towards obtaining the desired fixed-parameter algorithm for sGASP is to show that every YES-instance contains a solution which is *acyclic* – in particular, a solution with no cycles formed by interactions between activities and agent types (captured in terms of the *incidence graph* G of an assignment). This is proved via the identification of certain *compression steps* which can be applied on a solution in order to remove cycles.

Once we show that it suffices to focus on acyclic solutions, we branch over all acyclic incidence graphs (i.e., all acyclic edge sets of G); for each such edge set, we can reduce the problem of determining whether there exists an assignment realizing this edge set to a variant of SUBSET SUM embedded in a tree structure. The last missing piece is then to show that this problem, which we call TREE SUBSET SUM, is polynomial-time tractable; this is done via dynamic programming, whereas each step boils down to solving a simplified variant of SUBSET SUM.

Result 2. sGASP is W1-hard when parameterized by t

Our second result complements Result 1. As a crucial intermediate step towards Result 2, we obtain the W[1]-hardness of a variant of SUBSET SUM with three distinct “ingredients”:

1. **Partitioning:** items are partitioned into sets, and precisely one item must be selected from each set,
2. **Multidimensionality:** each item is a d -dimensional vector (d being the parameter) where the aim is to hit the target value for each component, and
3. **Simplicity:** each vector contains precisely one non-zero component.

We call this problem SIMPLE MULTIDIMENSIONAL PARTITIONED SUBSET SUM (SMPSS). Note that SMPSS is closely related to MULTIDIMENSIONAL SUBSET SUM (MSS), which (as one would expect) merely enhances SUBSET SUM via Ingredient 2. MSS has recently been used to establish W[1]-hardness for parameterizations of EDGE DISJOINT PATHS [13] and BOUNDED DEGREE VERTEX DELETION [12]. However, Ingredient 1 and especially Ingredient 3 are critical requirements for our reduction to work, and establishing the W[1]-hardness of SMPSS was the main challenge on the way towards the desired lower-bound result for sGASP. Since MSS has already been successfully used to obtain lower-bound results and SMPSS is a much more powerful tool in this regard, we believe that SMPSS will find applications in establishing lower bounds for other problems in the future.

Result 3. GASP is in XP when parameterized by t

This is the only XP result required for our complexity map, as it implies XP algorithms for sGASP parameterized by t and for GASP parameterized by t . We note that the techniques used to obtain Result 3 are disjoint from those used for Result 1; in particular, our first step is to reduce GASP parameterized by t to solving “XP-many” instances of sGASP parameterized by t . This is achieved by showing that once we know a “least preferred alternative” for every agent type that is active in an assignment, then the GASP instance becomes significantly easier – and, in particular, can be reduced to a (slightly modified version of) sGASP. It is interesting to note that the result provides a significant conceptual improvement over the known brute force algorithm for GASP parameterized by the number of agents which enumerates all possible assignments of agents to activities [16, Theorem 3] (see also [15]): instead of guessing an assignment for all agents, one merely needs to guess a least preferred alternative for every agent type.

The second part of our journey towards Result 3 focuses on obtaining an XP algorithm for sGASP parameterized by t . This algorithm has two components. Initially, we show that in this setting one can reduce sGASP to the problem of finding an assignment which is individually rational (i.e., without the stability condition) and satisfies some additional minor properties. To find such an assignment, we once again make use of SUBSET SUM: in particular, we develop an XP algorithm for the MPSS problem (i.e., SUBSET SUM enhanced by ingredients 1 and 2) and apply a final reduction from finding an individually rational assignment to MPSS.

Result 4. GASP is W1-hard when parameterized by $t + a$ **Result 5.** gGASP is W1-hard when parameterized by $t + a$ and the *vertex cover number* [11] of the network

The final two results are hardness reductions which represent the last pieces of the presented complexity map. Both are obtained via reductions from PARTITIONED CLIQUE (also called MULTICOLORED CLIQUE in the literature [2]), and both reductions essentially use $k + \binom{k}{2}$ activities whose sizes encode the vertices and edges forming a k -clique. The main challenge lies in the design of (a bounded number of) agent types whose preference lists ensure that the chosen vertices are indeed endpoints of the chosen edges. The reduction for gGASP then becomes even more involved, as it can only employ a limited number of connections between the agents in order to ensure that vertex cover of the network is bounded.

We note that Result 5 also followed up on previous work by Gupta, Roy, Saurabh and Zehavi [14], who showed that gGASP is fixed-parameter tractable parameterized by the number of activities if the network has constant treewidth. In this sense, our hardness result represents a substantial shift of the boundaries of (in)tractability: in addition to excluding fixed-parameter tractability when parameterizing by the number of activities and treewidth, it also rules out the use of agent types as a parameter and replaces treewidth by the more restrictive vertex cover number. An overview of our results is provided in Table 1.

2 Preliminaries

For an integer i , we let $[i] = \{1, 2, \dots, i\}$ and $[i]_0 = [i] \cup \{0\}$. We denote by \mathbb{N} the set of natural numbers, by \mathbb{N}_0 the set $\mathbb{N} \cup \{0\}$. For a set S and an integer k , we denote by S^k and 2^S the set of all k dimensional vectors over S and the set of all subsets of S , respectively.

■ **Table 1** Lower and upper bounds for sGASP, GASP, and gGASP parameterized by the number of agent types (t), with or without additionally parameterizing by the number of activities (a). In the case of gGASP, also the parameter vertex cover number (vc) of the network is considered. The numbers 1 to 5 in the upper index are used to identify results 1 to 5. Entries in bold are shown in this paper; previously known entries follow from the work of Gupta et al. [14].

	Parameter	Lower Bound	Upper Bound
sGASP	t	W[1]²	XP
GASP		W[1]	XP³
gGASP		paraNP	
sGASP	$t + a$		FPT¹
GASP		W[1]⁴	XP
gGASP		paraNP	
gGASP	$t + a + vc$	W[1]⁵	XP

We refer to the handbook by Diestel ([7]) for standard graph terminology. Due to space constraints, we also refer to the respective handbooks [8, 2] for standard terminology and basic notions in parameterized complexity. The *vertex cover number* of a graph G is the size of a minimum vertex cover of G .

2.1 Group Activity Selection

The task in the GROUP ACTIVITY SELECTION PROBLEM (GASP) is to compute a *stable assignment* π from a given set N of *agents* to a set A of *activities*, where each agent participates in at most one activity in A . The assignment π is (Nash) stable if and only if it is *individually rational* and no agent has an *NS-deviation* to any other activity (both of these stability rules are defined in the next paragraph). We use a dummy activity a_\emptyset to capture all those agents that do not participate in any activity in A and denote by A^* the set $A \cup \{a_\emptyset\}$. Thus, an assignment π is a mapping from N to A^* , and for an activity $a \in A$ we use $\pi^{-1}(a)$ to denote the set of agents assigned to a by π ; we set $|\pi^{-1}(a_\emptyset)| = 1$ if there is at least one agent assigned to a_\emptyset and 0 otherwise.

The set X of *alternatives* is defined as $X = (A \times [|N|]) \cup \{(a_\emptyset, 1)\}$. Each agent is associated with its own *preferences* defined on the set X . In the case of the standard GASP problem, an instance I is of the form $(N, A, (\succeq_n)_{n \in N})$ where each agent n is associated with a complete transitive preference relation (list) \succeq_n over the set X . An assignment π is *individually rational* if for every agent $n \in N$ it holds that if $\pi(n) = a$ and $a \neq a_\emptyset$, then $(a, |\pi^{-1}(a)|) \succeq_n (a_\emptyset, 1)$ (i.e., n weakly prefers staying in a over moving to a_\emptyset). An agent n where $\pi(n) = a$ is defined to have an *NS-deviation* to a different activity a' in A if $(a', |\pi^{-1}(a')| + 1) \succ_n (a, |\pi^{-1}(a)|)$ (i.e., n prefers moving to an activity a' over staying in a). The task in GASP is to compute a stable assignment.

gGASP is defined analogously to GASP, however where one is additionally given a set L of *links* $L \subseteq \{\{n, n'\} \mid n, n' \in N \wedge n \neq n'\}$ between the agents on the input; specifically, L can be viewed as a set of undirected edges and (N, L) as a simple undirected graph. In gGASP, the task is to find an assignment π which is not only stable but also connected; formally, for every $a \in A$ the set of agents $\pi^{-1}(a)$ induces a connected subgraph of (N, L) . Moreover, an agent $n \in N$ only has an NS-deviation to some activity $a \neq \pi(n)$ if (in addition to the conditions for NS-deviations defined above) n has an edge to at least one agent in $\pi^{-1}(a)$.

In sGASP, an instance I is of the form $(N, A, (P_n)_{n \in N})$, where each agent has an *approval set* $P_n \subseteq X \setminus \{(a_\emptyset, 1)\}$ of preferences (instead of an ordered preference list). We denote by $P_n(a)$ the set $\{i \mid (a, i) \in P_n\}$ for an activity $a \in A$. An assignment $\pi : N \rightarrow A^*$ is said to be *individually rational* if every agent $n \in N$ satisfied the following: if $\pi(n) = a$ and $a \neq a_\emptyset$, then $|\pi^{-1}(a)| \in P_n(a)$. Further, an agent $n \in N$ where $\pi(n) = a_\emptyset$, is said to have an *NS-deviation* to an activity a in A if $(|\pi^{-1}(a)| + 1) \in P_n(a)$.

We now introduce the notions and definitions required for our main parameter of interest, the “number of agent types”. We say that two agents n and n' in N have the same *agent type* if they have the same preferences. To be specific, $P_n = P_{n'}$ for sGASP and $\succeq_n = \succeq_{n'}$ for GASP and gGASP. In the case of sGASP and GASP n and n' are indistinguishable, while in gGASP n and n' can still have different links to other agents. For a subset $N' \subseteq N$, we denote by $T(N')$ the set of agent types occurring in N' . Note that this notation requires that the instance is clear from the context. If this is not the case then we denote by $T(I)$ the set $T(N)$ if N is the set of agents for the instance I of sGASP, GASP, or gGASP.

For every agent type $t \in T(I)$, we denote by N_t the subset of N containing all agents of type t ; observe that $\{N_t \mid t \in T(I)\}$ forms a partition of N . For an agent type $t \in T(I)$ we denote by P_t (sGASP) or \succeq_t (GASP) the preference list assigned to all agents of type t and we use $P_t(a)$ (for an activity $a \in A$) to denote P_t restricted to activity a , i.e., $P_t(a)$ is equal to $P_n(a)$ for any agent n of type t . For an assignment $\pi : N \rightarrow A^*$, $t \in T(I)$, and $a \in A$ we denote by $\pi_{t,a}$ the set $\{n \mid n \in N_t \wedge \pi(n) = a\}$ and by π_t the set $\bigcup_{a \in A} \pi_{t,a}$. Further, $\pi(t)$ is the set of all activities that have at least one agent of type t participating in it. We say that π is a perfect assignment for some agent type $t \in T(I)$ if $\pi(n) \neq a_\emptyset$ for every $n \in N_t$. We denote by $\text{PE}(I, \pi)$ the subset of $T(I)$ consisting of all agent types that are perfectly assigned by π , and say that π is a *perfect assignment* if $\text{PE}(I, \pi) = T(I)$.

One notion that will appear through the paper is that of *compatibility*: for a subset $Q \subseteq T(I)$, we say that π is *compatible* with Q if $\text{PE}(I, \pi) = Q$. We conclude this section with a technical lemma which provides a preprocessing procedure that will be used as a basic tool for obtaining our algorithmic results. In particular, Lemma 1 allows us to reduce the problem of computing a stable assignment for a sGASP instance compatible with Q to the problem of finding an individually rational assignment.

► **Lemma 1.** *Let $I = (N, A, (P_n)_{n \in N})$ be an instance of sGASP and $Q \subseteq T(I)$. Then in time $\mathcal{O}(|N|^2|A|)$ one can compute an instance $\gamma(I, Q) = (N, A, (P'_n)_{n \in N})$ and $A_{\neq \emptyset}(I, Q) \subseteq A$ with the following property: for every assignment $\pi : N \rightarrow A^*$ that is compatible with Q , it holds that π is stable for I if and only if π is individually rational for $\gamma(I, Q)$ and $\pi^{-1}(a) \neq \emptyset$ for every $a \in A_{\neq \emptyset}(I, Q)$.*

3 Subset Sum Machinery

In this section we introduce the Subset Sum machinery required for our algorithms and lower bound results. In particular, we introduce three variants of SUBSET SUM, obtain algorithms for two of them, and provide a W[1]-hardness result for the third.

Tree Subset Sum. Here we introduce a useful generalization of SUBSET SUM, for which we obtain polynomial-time tractability under the assumption that the input is encoded in unary. Intuitively, our problem asks us to assign values to edges while meeting a simple criterion on the values of edges incident to each vertex.

TREE SUBSET SUM (TSS)

Input: A vertex-labeled undirected tree T with labeling function $\lambda : V(T) \rightarrow 2^{\mathbb{N}}$.
 Question: Is there an assignment $\alpha : E(T) \rightarrow \mathbb{N}$ such that for every $v \in V(T)$ it holds that $\sum_{e \in E(T) \wedge v \in e} \alpha(e) \in \lambda(v)$.

Let us briefly comment on the relationship of TSS with SUBSET SUM. Recall that given a set S of natural numbers and a natural number t , the SUBSET SUM problem asks whether there is a subset S' of S such that $\sum_{s \in S'} s = t$. One can easily construct a simple instance (G, λ) of TSS that is equivalent to a given instance (S, t) of SUBSET SUM as follows. G consists of a star having one leaf l_s for every $s \in S$ with $\lambda(l_s) = \{0, s\}$ and $\lambda(c) = \{t\}$ for the center vertex c of the star. Given this simple reduction from SUBSET SUM to TSS it becomes clear that TSS is much more general than SUBSET SUM. In particular, instead of a star TSS allows for the use of an arbitrary tree structure and moreover one can use arbitrary subsets of natural numbers to specify the constraints on the vertices. The above reduction in combination with the fact that SUBSET SUM is weakly NP-hard implies that TSS is also weakly NP-hard. In the remainder of this paragraph we will show that TSS (like SUBSET SUM) can be solved in polynomial-time if the input is given in unary. This will later be used to obtain Result 1 (in Section 4).

Let $I = (T, \lambda)$ be an instance of TSS. We denote by $\max(I)$ the value of the maximum number occurring in any vertex label. The main idea behind our algorithm for TSS is to apply leaf-to-root dynamic programming. In order to execute our dynamic programming procedure, we will need to solve a special case of TSS which we call PARTITIONED SUBSET SUM; this is the problem that will later arise when computing the dynamic programming tables for TSS. In the PARTITIONED SUBSET SUM problem one is given a target set R of natural numbers and ℓ source sets S_1, \dots, S_ℓ of natural numbers and the aim is to compute the set S of all natural numbers s such that there are s_1, \dots, s_ℓ , where $s_i \in S_i$ for every i with $1 \leq i \leq \ell$, satisfying $(\sum_{1 \leq i \leq \ell} s_i) + s \in R$.

► **Lemma 2.** *An instance $I = (T, \lambda)$ of TSS can be solved in time $\mathcal{O}(|V(T)|^2 \cdot \max(I)^2)$.*

Multidimensional Partitioned Subset Sum. Our second generalization of SUBSET SUM is a multi-dimensional variant of the problem that allows to separate the input set of numbers into several groups, and restricts the solution to take at most 1 vector from each group. For technical reasons, we will only search for solutions of size at most r .

MULTIDIMENSIONAL PARTITIONED SUBSET SUM (MPSS)

Input: $k \in \mathbb{N}$, $r \in \mathbb{N}_0$, and a family $\mathcal{P} = \{P_1, \dots, P_l\}$ of sets of vectors over \mathbb{N}_0^k .
 Question: Compute the set of all vectors $\bar{t} \in \{0, \dots, r\}^k$ such that there are $\bar{p}_1, \dots, \bar{p}_l$ with $\bar{p}_i \in P_i$ for every i with $1 \leq i \leq l$ such that $\sum_{1 \leq i \leq l} \bar{p}_i = \bar{t}$.

It is easy to see that SUBSET SUM is a special case of MPSS: given an instance of SUBSET SUM, we can create an equivalent instance of MPSS by setting r to a sufficiently large number and simply making each group P_i contain two vectors: the all-zero vector and the vector that is equal to the i -th number of the SUBSET SUM instance in all entries. The following algorithm is used as a subprocedure for Result 3 (in Section 6).

► **Lemma 3.** *An instance $I = (k, r, \mathcal{P})$ of MPSS can be solved in time $\mathcal{O}(|I| \cdot r^k)$.*

Simple Multidimensional Partitioned Subset Sum. Here, we are interested in a much more restrictive version of MPSS, where all vectors (apart from the target vector) are only allowed to have at most one non-zero component. Surprisingly, we show that the $W[1]$ -hardness of the previously studied MULTIDIMENSIONAL SUBSET SUM problem [13, 12] carries over to this more restrictive variant using an intricate and involved reduction. This result forms the main ingredient needed for our Result 2 (provided in Section 5). To formalize our problem, we say that a set P of vectors in \mathbb{N}_0^d is *simple* if each vector in P has at most one non-zero component and the values of the non-zero components for any two distinct vectors in P are distinct.

SIMPLE MULTIDIMENSIONAL PARTITIONED SUBSET SUM (SMPSS)	
Input:	$d \in \mathbb{N}$, $\bar{t} \in \mathbb{N}_0^d$, and a family $\mathcal{P} = \{P_1, \dots, P_l\}$ of simple sets of vectors in \mathbb{N}_0^d .
Parameter:	d .
Question:	Are there vectors $\bar{p}_1, \dots, \bar{p}_l$ with $\bar{p}_i \in P_i$ for every i with $1 \leq i \leq l$ such that $\sum_{1 \leq i \leq l} \bar{p}_i = \bar{t}$.

► **Theorem 4.** *SMPSS is strongly $W[1]$ -hard.*

Proof Sketch. We will employ a parameterized reduction from the PARTITIONED CLIQUE problem, which is well-known to be $W[1]$ -complete [20]. In PARTITIONED CLIQUE we are given an integer k along with a graph G whose vertex set V is partitioned into k given independent sets V_1, \dots, V_k , and are asked to decide whether G contains a k -clique. We denote by $E_{i,j}$ the set of edges of G that have one endpoint in V_i and one endpoint in V_j and we assume w.l.o.g. that $|V_i| = \{v_1^i, \dots, v_n^i\} = n$ and $|E_{i,j}| = m$ for every i and j with $1 \leq i < j \leq k$ (see the standard textbooks for a justification of these assumptions [2, 8]).

Given an instance (G, k) of PARTITIONED CLIQUE with partition V_1, \dots, V_k , we construct an equivalent instance $I = (d, \bar{t}, \mathcal{P})$ of SMPSS in polynomial time, where $d = k(k-1) + \binom{k}{2}$ and $|\mathcal{P}| = \binom{k}{2} + nk(2k-3)$. We will also make use of the following notation. For i and j with $1 \leq i \leq k$ and $1 \leq j < k$, we denote by $\text{indJ}(i, j)$ the j -th smallest number in $[k] \setminus \{i\}$ and we denote by $\text{indMin}(i)$ and $\text{indMax}(i)$ the numbers $\text{indJ}(i, 1)$ and $\text{indJ}(i, k-1)$, respectively.

We assign to every vertex v of G a unique number $\mathcal{S}(v)$ from a Sidon sequence \mathcal{S} of length $|V(G)|$ [10]. A *Sidon sequence* is a sequence of natural numbers such that the sum of each pair of numbers is unique; it can be shown that it is possible to construct such sequences whose maximum value is bounded by a polynomial in its length [1, 10].

To simplify the description of I , we will introduce names and notions to identify both components of vectors and sets in \mathcal{P} . Every vector in I has the following components:

- For every i and j with $1 \leq i, j \leq k$ and $i \neq j$, the *vertex component* $c_V^i(j)$. We set $\bar{t}[c_V^i(j)]$ to:
 - $n^6 + n^4$ if $j = \text{indMin}(i)$,
 - $(n-1)n^8 + n^6 + n^4 + \sum_{\ell=1}^n (\ell + \ell n^2)$ if $j > \text{indMin}(i)$ and $j < \text{indMax}(i)$, and
 - $(n-1)n^8 + n^6 + \sum_{\ell=1}^n \ell$, otherwise.
- For every i and j with $1 \leq i < j \leq k$, the *edge component* $c_E(i, j)$ with $\bar{t}[c_E(i, j)] = \sum_{v \in V_i \cup V_j} \mathcal{S}(v)$.

Note that the total number of components d is equal to $k(k-1) + \binom{k}{2}$ and that for every i with $1 \leq i \leq k$, there are $k-1$ vertex components, i.e., the components $c_V^i(\text{indJ}(i, 1)), \dots, c_V^i(\text{indJ}(i, k-1))$, which intuitively have the following tasks. The first component, i.e., the component $c_V^i(\text{indJ}(i, 1))$ identifies a vertex $v \in V_i$ that should be part of a k -clique in G . Moreover, every component $c_V^i(\text{indJ}(i, j))$ (including the first component), is also responsible for: (1) Signalling the choice of the chosen vertex $v \in V_i$ to the next component,

i.e., the component $c_V^i(\text{indJ}(i, j + 1))$ and (2) Signalling the choice of the vertex $v \in V_i$ to the component $c_E(i, j)$ that will then verify that there is an edge between the vertex chosen for V_i and the vertex chosen for V_j . This interplay between the components will be achieved through the sets of vectors in \mathcal{P} that will be defined and explained next.

■ **Table 2** An illustration of the vectors contained in the sets $P_{EV}^1(2, \ell), \dots, P_{EV}^1(4, \ell)$ and $P_V^1(2, \ell), \dots, P_V^1(3, \ell)$. For example the column for the set $P_{EV}^1(2, \ell)$ shows that the set contains two vectors, one whose only non-zero component $c_V^1(2)$ has the value $n^6 + \ell$ and a second one whose only non-zero component $c_E(1, 2)$ and has the value $\mathcal{S}(v_\ell^1)$. The last column provides the value for the target vector for the component given by the row. Finally, the value Z is equal to $(n - 1)n^8 + n^6 + \sum_{\ell=1}^n (\ell)$.

	$P_{EV}^1(2, \ell)$	$P_V^1(2, \ell)$	$P_{EV}^1(3, \ell)$	$P_V^1(3, \ell)$	$P_{EV}^1(4, \ell)$	\bar{t}
$c_V^1(2)$	$n^6 + \ell$	$n^4 - \ell$				$n^6 + n^4$
$c_V^1(3)$		$n^8 + \ell + \ell n^2$	$n^6 + \ell$	$n^4 + \ell n^2$		$Z + n^4 + \sum_{\ell=1}^n (\ell n^2)$
$c_V^1(4)$				$n^8 + \ell$	$n^6 + \ell$	Z
$c_E(1, 2)$	$\mathcal{S}(v_\ell^1)$					$\sum_{v \in V_1 \cup V_2} \mathcal{S}(v)$
$c_E(1, 3)$			$\mathcal{S}(v_\ell^1)$			$\sum_{v \in V_1 \cup V_3} \mathcal{S}(v)$
$c_E(1, 4)$					$\mathcal{S}(v_\ell^1)$	$\sum_{v \in V_1 \cup V_4} \mathcal{S}(v)$

■ **Table 3** An illustration of the vectors contained in the sets $P_{EV}^i(j, \ell)$, $P_{EV}^j(i, \ell)$, and $P_E(i, j)$ and their interplay with the components $c_V^i(j)$, $c_V^j(i)$, and $c_V(i, j)$. For the conventions used in the table please refer to Table 2. Additionally, note that the column for $P_E(i, j)$ indicates that the set contains one vector for every edge $\{v, u\} \in E_{i,j}$, whose only non-zero component $c_E(i, j)$ has the value $\mathcal{S}(v) + \mathcal{S}(u)$.

	$P_{EV}^i(j, \ell)$	$P_{EV}^j(i, \ell)$	$P_E(i, j)$	\bar{t}
$c_V^i(j)$	$n^6 + \ell$			
$c_V^j(i)$		$n^6 + \ell$		
$c_E(i, j)$	$\mathcal{S}(v_\ell^i)$	$\mathcal{S}(v_\ell^j)$	$\{\mathcal{S}(v) + \mathcal{S}(u) \mid \{v, u\} \in E_{i,j}\}$	$\sum_{v \in V_i \cup V_j} \mathcal{S}(v)$

\mathcal{P} consists of the following sets, which are illustrated in Table 2 and 3:

- For every i, j' , and ℓ with $1 \leq i \leq k$, $1 \leq j' \leq k - 2$, and $1 \leq \ell \leq n$, the *vertex set* $P_V^i(j, \ell)$, where $j = \text{indJ}(i, j')$, containing two vectors $\bar{v}_{i,j,\ell}^+$ and $\bar{v}_{i,j,\ell}^-$ defined as follows:
 - if $j' = 1$, then $\bar{v}_{i,j,\ell}^+[c_V^i(j)] = n^4 - \ell$ and $\bar{v}_{i,j,\ell}^-[c_V^i(\text{indJ}(i, j' + 1))] = n^8 + \ell + \ell n^2$ or
 - if $1 < j' < k - 2$, then $\bar{v}_{i,j,\ell}^+[c_V^i(j)] = n^4 + \ell n^2$ and $\bar{v}_{i,j,\ell}^-[c_V^i(\text{indJ}(i, j' + 1))] = n^8 + \ell + \ell n^2$ or
 - if $j' = k - 2$, then $\bar{v}_{i,j,\ell}^+[c_V^i(j)] = n^4 + \ell n^2$ and $\bar{v}_{i,j,\ell}^-[c_V^i(\text{indJ}(i, j' + 1))] = n^8 + \ell$.
 We denote by $P_V^i(j)$, $P_{V^+}^i(j)$, and $P_{V^-}^i(j)$ the sets $\bigcup_{\ell=1}^n (P_V^i(j, \ell))$, $P_V^i(j) \cap \{\bar{v}_{i,j,\ell}^+ \mid 1 \leq \ell \leq n\}$, and $P_V^i(j) \setminus P_{V^+}^i(j)$, respectively.
- For every i, j , and ℓ with $1 \leq i, j \leq k$, $i \neq j$, and $1 \leq \ell \leq n$, the *vertex incidence set* $P_{EV}^i(j, \ell)$, which contains the two vectors $\bar{a}_{i,j,\ell}^+$ and $\bar{a}_{i,j,\ell}^-$ such that $\bar{a}_{i,j,\ell}^+[c_V^i(j)] = n^6 + \ell$ and $\bar{a}_{i,j,\ell}^-[c_E(i, j)] = \mathcal{S}(v_\ell^i)$. We denote by $P_{EV}^i(j)$, $P_{EV^+}^i(j)$, and $P_{EV^-}^i(j)$ the sets $\bigcup_{\ell=1}^n (P_{EV}^i(j, \ell))$, $P_V^i(j) \cap \{\bar{a}_{i,j,\ell}^+ \mid 1 \leq \ell \leq n\}$, and $P_{EV}^i(j) \setminus P_{EV^+}^i(j)$, respectively.
- For every i, j with $1 \leq i < j \leq k$, the *edge set* $P_E(i, j)$, which for every $e = \{v, u\} \in E_{i,j}$ contains the vector \bar{e} such that $\bar{e}[c_E(i, j)] = \mathcal{S}(v) + \mathcal{S}(u)$; note that $P_E(i, j)$ is indeed a simple set, because \mathcal{S} is a Sidon sequence.

Note that altogether there are $nk(k - 2) + \binom{k}{2} + nk(k - 1) = \binom{k}{2} + nk(2k - 3)$ sets in \mathcal{P} .

Informally, the two vectors $\bar{v}_{i,j,\ell}^+$ and $\bar{v}_{i,j,\ell}^-$ in $P_V^i(j, \ell)$ represent the choice of whether or not the vertex v_ℓ^i should be included in a k -clique for G , i.e., if a solution for I chooses $v_{i,j,\ell}^+$ then v_ℓ^i should be part of a k -clique and otherwise not. The component $c_V^i(j)$, more specifically the value for $\bar{t}[c_V^i(j)]$, now ensures that a solution can choose at most one such vector in $P_{V^+}^i(j)$. Moreover, the fact that all but one of the vectors $\bar{v}_{i,j,1}^-, \dots, \bar{v}_{i,j,n}^-$ need to be chosen by a solution for I signals the choice of the vertex for V_i to the next component, i.e., either the component $c_V^i(j+1)$ if $j+1 \neq i$ or the component $c_V^i(j+2)$ if $j+1 = i$. Note that we only need $k-2$ sets $P_V^i(j)$ for every i , because we need to copy the vertex choice for V_i to only $k-1$ components. A similar idea underlies the two vectors $\bar{a}_{i,j,\ell}^+$ and $\bar{a}_{i,j,\ell}^-$ in $P_{EV}^i(j, \ell)$, i.e., again the component $c_V^i(j)$ ensures that $\bar{a}_{i,j,\ell}^+$ can be chosen for only one of the sets $P_{EV}^i(j, 1), \dots, P_{EV}^i(j, n)$ and $\bar{a}_{i,j,\ell}^-$ must be chosen for all the remaining ones. Note that the component $c_V^i(j)$ now also ensures that the choice made for the sets in $P_V^i(j)$ is the same as the choice made for the sets in $P_{EV}^i(j)$. Moreover, the choice made for the sets in $P_{EV}^i(j)$ is now propagated to the component $c_E(i, j)$ (instead of the next vertex component). Finally, the vectors in the set $P_E(i, j)$ represent the choice of the edge used in a k -clique between V_i and V_j and the component $c_E(i, j)$ ensures that only an edge, whose endpoints are the two vertices signalled by the sets $P_{EV}^i(j)$ and $P_{EV}^j(i)$ can be chosen. \blacktriangleleft

4 Result 1: Fixed-Parameter Tractability of sGASP

In this section we will establish that sGASP is FPT when parameterized by the number of agent types and the number of activities by proving Theorem 5.

► **Theorem 5.** sGASP can be solved in time $\mathcal{O}(2^{|T(N)| \cdot (1+|A|)} \cdot ((|N| + |A|)|N|)^2)$.

Let $I = (N, A, (P_n)_{n \in N})$ be a sGASP instance and let $\pi : N \rightarrow A^*$ be an assignment of agents to activities. We denote by $G_I(\pi)$ the incidence graph between $T(N)$ and A , which is defined as follows. $G_I(\pi)$ has vertices $T(N) \cup A$ and contains an edge between an agent type $t \in T(N)$ and an activity $a \in A$ if $\pi_{t,a} \neq \emptyset$. We say that π is *acyclic* if $G_I(\pi)$ is acyclic.

Our first aim towards the proof of Theorem 5 is to show that if I has a stable assignment, then it also has an acyclic stable assignment (Lemma 7). We will then show in Lemma 9 that finding a stable assignment whose incidence graph is equal to some given acyclic pattern graph can be achieved in polynomial-time via a reduction to the TSS problem (see Lemma 2). Since the number of (acyclic) pattern graphs is bounded in our parameters, we can subsequently solve sGASP by enumerating all acyclic pattern graphs and checking for each of them whether there is an acyclic solution matching the selected pattern.

A crucial notion towards showing that it is sufficient to consider only acyclic solutions is the notion of (strict) compression. We say that an assignment τ is a *compression* of π if it satisfies the following conditions:

- (C1) for every $t \in T(N)$ it holds that $|\pi_t| = |\tau_t|$,
- (C2) for every $a \in A$ it holds that $|\pi^{-1}(a)| = |\tau^{-1}(a)|$, and
- (C3) for every $a \in A$ it holds that the set of agent types τ assigns to a is a subset of the agent types π assigns to a .

Intuitively, an assignment τ is a compression of π if it maintains all the properties required to preserve stability and compatibility with a given subset $Q \subseteq T(N)$. We note that condition (C3) can be formalized as $T(\pi^{-1}(a)) \supseteq T(\tau^{-1}(a))$. The following lemma shows that every assignment that is not acyclic admits a compression.

► **Lemma 6.** Let $\pi : N \rightarrow A^*$ be an assignment for I . Then there exists an acyclic assignment π' that compresses π .

The next lemma provides the first cornerstone for our algorithm by showing that it is sufficient to consider only acyclic solutions. Intuitively, it is a consequence of Lemma 6 along with the observation that compression preserves stability and individual rationality.

► **Lemma 7.** *If I has a stable assignment, then I has an acyclic stable assignment.*

Our next step is the introduction of terminology related to the pattern graphs mentioned at the beginning of this section. Let G be a bipartite graph with bi-partition $\{T(N), A\}$. We say that G *models* an assignment $\pi : N \rightarrow A^*$ if $G_I(\pi) = G$; in this sense every such bipartite graph can be seen as a pattern (or model) for assignments. For a subset $Q \subseteq T(N)$ we say that G is *compatible* with Q if every vertex in Q and every vertex in $A_{\neq \emptyset}(I, Q)$ (recall the definition of $A_{\neq \emptyset}(I, Q)$ given in Lemma 1) has at least one neighbor in G ; note that if G is compatible with Q then any assignment π modeled by G satisfies $\tau^{-1}(a) \neq \emptyset$ for every $a \in A_{\neq \emptyset}(I, Q)$. Intuitively, the graph G captures information about which types of agents are mapped to which activities (without specifying numbers), while Q captures information about which agent types are perfectly (i.e., “completely”) assigned.

Let $Q \subseteq T(N)$ and let G be a bipartite graph with bi-partition $\{T(N), A\}$ that is compatible with Q . The following simple lemma shows that, modulo compatibility requirements, finding a stable assignment for I can be reduced to finding an individually rational assignment for $\gamma(I, Q)$ (recall the definition of $\gamma(I, Q)$ given in Lemma 1).

► **Lemma 8.** *Let $Q \subseteq T(N)$ and let G be a bipartite graph with bi-partition $\{T(N), A\}$ that is compatible with Q . Then for every assignment $\pi : N \rightarrow A^*$ modeled by G and compatible with Q , π is stable for I if and only if π is individually rational for $\gamma(I, Q)$.*

The next lemma forms (along with Lemma 7) the core component for our proof.

► **Lemma 9.** *Let $Q \subseteq T(N)$ and let G be an acyclic bipartite graph with bi-partition $\{T(N), A\}$ that is compatible with Q . Then one can decide in time $\mathcal{O}((|N| + |A|)^2 |N|^2)$ whether I has a stable assignment which is modeled by G and compatible with Q .*

We now have all the ingredients needed to establish Theorem 5 (\star).

5 Result 2: Lower Bound for sGASP

In this section we complement Theorem 5 by showing that if we drop the number of activities in the parameterization, then sGASP becomes W[1]-hard. We achieve this via a parameterized reduction from SMPSS that we have shown to be strongly W[1]-hard in Theorem 4.

► **Theorem 10.** *sGASP is W[1]-hard parameterized by the number of agent types.*

6 Result 3: XP Algorithms for sGASP and GASP

In this section, we present our XP algorithm for GASP parameterized by the number of agent types. In order to obtain this result, we observe that the stability of an assignment for GASP can be decided by only considering the stability of agents that are assigned to a “minimal alternative” w.r.t. their type. We then show that once one guesses (i.e., branches over) a minimal alternative for every agent type, the problem of finding a stable assignment for GASP that is compatible with this guess can be reduced to the problem of finding a perfect and individual rational assignment for a certain instance of sGASP, where one additionally requires that certain activities are assigned to at least one agent. Our first task will hence be to obtain an XP algorithm which can find such a perfect and individually rational assignment for sGASP. To that end, we obtain Lemma 11, which allows us to find certain individually rational assignments in sGASP instances and forms a core part of our XP algorithm for GASP.

48:12 Group Activity Selection with Few Agent Types

► **Lemma 11.** *Let $I = (N, A, (P_n)_{n \in N})$ be an instance of sGASP, $Q \subseteq T(N)$, and $A_{\neq \emptyset} \subseteq A$. Then one can decide in time $\mathcal{O}(|A| \cdot (|N|)^{|T(N)|})$ whether I has an individual rational assignment π that is compatible with Q such that $\pi^{-1}(a) \neq \emptyset$ for every $a \in A_{\neq \emptyset}$.*

As a secondary result, we can already obtain an XP algorithm for sGASP parameterized by the number of agent types, which may also be of independent interest, as the obtained running time is strictly better than that of the algorithm obtained for the more general GASP.

► **Theorem 12.** *An instance $I = (N, A, (P_n)_{n \in N})$ of sGASP can be solved in time $|A| \cdot |N|^{\mathcal{O}(|T(N)|)}$.*

Our next aim is to use Lemma 11 to obtain an XP algorithm for GASP. To simplify the presentation of our algorithm, we start by introducing the notion of an NS*-deviations that combines and unifies individual rationality and NS-deviations. Namely, let $I = (N, A, (\succeq_n)_{n \in N})$ be a GASP instance, $\pi : N \rightarrow A^*$ be an assignment, and $n \in N$. We say that n has an NS*-deviation to an activity $a' \in A^* \setminus \{\pi(n)\}$ if $(a', |\pi^{-1}(a')| + 1) \succ_{T(n)} (a, |\pi^{-1}(a)|)$. In order to deal with the case that $a' = a_\emptyset$, we let $(a_\emptyset, i + 1)$ stand for $(a_\emptyset, 1)$ for every i .

► **Observation 13.** An assignment π for I is stable if and only if no agent $n \in N$ has an NS*-deviation to any activity in $A^* \setminus \{\pi(n)\}$.

Let I and π be as above and let $t \in T(I)$. We denote by π_t^* the set of activities π_i if t is perfectly assigned by π and $\pi_t \cup \{a_\emptyset\}$, otherwise. We say an activity $a \in \pi_t^*$ is *minimal with respect to t* if $(a', |\pi^{-1}(a')|) \succeq_t (a, |\pi^{-1}(a)|)$ for each $a' \in \pi_t^*$ and we address the alternative $(a, |\pi^{-1}(a)|)$ as a *minimal alternative* with respect to t .

The following lemma uses Observation 13 and allows us to characterize the stability condition of an assignment in terms of minimal activities for each agent type.

► **Lemma 14.** *An assignment π for I is stable if and only if for each $t \in T(N)$ and each $a \in A^* \setminus \{a_m\}$, it holds that $(a_m, |\pi^{-1}(a_m)|) \succeq_t (a, |\pi^{-1}(a)| + 1)$, where a_m is a minimal activity w.r.t. t .*

The next theorem now employs the above lemma to construct an instance I' of sGASP together with a subset $A_{\neq \emptyset}$ of activities such that for every function $f_{\min} : T(I) \rightarrow X$ (or in other words for every guess of minimal alternatives in an assignment), it holds that I has a stable assignment such that $f_{\min}(t)$ is a minimal alternative w.r.t. t for every $t \in T(I)$ if and only if I' has a perfect and individual rational assignment π such that $\pi^{-1}(a) \neq \emptyset$ for every $a \in A_{\neq \emptyset}$. For brevity, we will say that an assignment π is *compatible with f_{\min}* if and only if $f_{\min}(t)$ is a minimal alternative w.r.t. t for every $t \in T(I)$.

► **Theorem 15.** *Let $I = (N, A, (\succeq_n)_{n \in N})$ be an instance of GASP and let $f_{\min} : T(N) \rightarrow X$, which informally represents a guess of a minimal alternative for every agent type. Then one can in time $\mathcal{O}(|N|^2 |A|)$ construct an instance $I' = (N, A \cup \{a_\emptyset\}, (P_n)_{n \in N})$ of sGASP together with a subset $A_{\neq \emptyset}$ of activities such that $|T(I')| \leq 2|T(I)|$ and I has a stable assignment compatible with $f_{\min}(t)$ if and only if I' has a perfect individual rational assignment π with $\pi^{-1}(a) \neq \emptyset$ for every $a \in A_{\neq \emptyset}$.*

We now have all the ingredients needed to prove the main result of this section.

► **Theorem 16.** *An instance $I = (N, A, (\succeq_n)_{n \in N})$ of GASP can be solved in time $(|A| \cdot |N|)^{\mathcal{O}(|T(I)|)}$.*

Proof Sketch. Given an instance $I = (N, A, (\succeq_n)_{n \in N})$ of GASP, the algorithm enumerates all of the at most $(|A| \cdot |N|)^{|T(I)|}$ possible functions f_{\min} and for each such function f_{\min} the algorithm uses Theorem 15 to construct the instance $I' = (N, A \cup \{a_\phi\}, (P_n)_{n \in N})$ of sGASP with $|T(I')| \leq |T(I)|$ together with the set $A_{\neq \emptyset}$ of activities in time $\mathcal{O}(|N|^2|A|)$. It then uses Lemma 11 to decide whether I' has a perfect individual rational assignment π_1 such that $\pi_1^{-1}(a) \neq \emptyset$ for every $a \in A_{\neq \emptyset}$ in time $\mathcal{O}((|A| + 1)(|N|)^{|T(I')|}) = \mathcal{O}((|A| + 1)(|N|^{2|T(I)|}))$. If this is true for at least one of the functions f_{\min} , the algorithm returns that I has a solution, otherwise the algorithm correctly returns that I has no solution. ◀

7 Results 4 and 5: Two Lower Bounds

Our next result shows that GASP is unlikely to be fixed-parameter tractable parameterized by both the number of activities (a) and the number of agent types (t).

► **Theorem 17.** *GASP is $W[1]$ -hard parameterized by $t + a$.*

Since GASP and gGASP are equivalent on complete networks the above hardness result clearly also applies to gGASP. However, to our surprise, the hardness does even hold if we additionally parameterize gGASP with the vertex cover number (vc) of the network.

► **Theorem 18.** *gGASP is $W[1]$ -hard parameterized by $t + a + vc$.*

8 Conclusion

We obtained a comprehensive picture of the parameterized complexity of Group Activity Selection problems parameterized by the number of agent types, both with and without the number of activities as an additional parameter. Our positive results suggest that using the number of agent types is a highly appealing parameter for GASP and its variants; indeed, this parameter will often be much smaller than the number of agents due to the way preference lists are collected or estimated (as also argued in initial work on GASP [6]). For instance, in the large-scale event management setting of GASP (or sGASP), one would expect that preference lists for event participants are collected via simple questionnaires – and so the number of agent types would remain fairly small regardless of the size of the event.

We believe that the techniques used to obtain the presented results, and especially the Subset Sum tools developed to this end, are of broad interest to the algorithms community. For instance, MULTIDIMENSIONAL SUBSET SUM (MSS) has been used as a starting point for $W[1]$ -hardness reductions in at least two different settings over the past year [13, 12], but the simple and partitioned variant of the problem (i.e., SMPSS) is much more restrictive and hence forms a strictly better starting point for any such reductions in the future. This is also reflected in our proof of the $W[1]$ -hardness of SMPSS, which is *significantly* more involved than the analogous result for MSS. Likewise, we expect that the developed algorithms for TREE SUBSET SUM and MULTIDIMENSIONAL PARTITIONED SUBSET SUM may find applications as subroutines for (parameterized and/or classical) algorithms in various settings.

Note that there is now an almost complete picture of the complexity of Group Activity Selection problems w.r.t. any combination of the parameters number of agents, number of activities, and number of agent types (see also Table 1). There is only one piece missing, namely, the parameterized complexity of sGASP parameterized by the number of agents, which we resolve for completeness with the following theorem.

► **Theorem 19.** *sGASP is fixed-parameter tractable parameterized by the number of agents.*

Proof. Let $I = (N, A, (P_n)_{n \in N})$ be a sGASP instance. The main idea behind the algorithm is to guess (i.e., branch over) the set M_\emptyset of agents that are assigned to a_\emptyset as well as a partition \mathcal{M} of the remaining agents, i.e., the agents in $N \setminus M_\emptyset$, and then check whether there is a stable assignment π for I such that:

(P1) $\pi^{-1}(a_\emptyset) = M_\emptyset$ and

(P2) $\{\pi^{-1}(a) \mid a \in A\} \setminus \{\emptyset\} = \mathcal{M}$, i.e., \mathcal{M} corresponds to the grouping of agents into activities by π .

Since there are at most n^n possibilities for M_\emptyset and \mathcal{M} and those can be enumerated in time $\mathcal{O}(n^n)$, it remains to show how to decide whether there is a stable assignment for I satisfying (P1) and (P2) for any given M_\emptyset and \mathcal{M} . Towards showing this, we first consider the implications for a stable assignment resulting from assigning the agents in M_\emptyset to a_\emptyset . Namely, let P'_n for every $n \in N$ be the approval set obtained from P_n after removing all alternatives (a, i) such that $i \neq 0$ and there is an agent $n_\emptyset \in M_\emptyset$ with $(a, i+1) \in P_{n_\emptyset}$. Moreover, let $A_{\neq \emptyset}$ be the set of all activities that cannot be left empty if the agents in M_\emptyset are assigned to a_\emptyset , i.e., the set of all activities such that there is an agent $n_\emptyset \in M_\emptyset$ with $(a, 1) \in P_{n_\emptyset}$. Now consider a set $M \in \mathcal{M}$, and observe that the set A_M of activities that the agents in M can be assigned to in any stable assignment satisfying (P1) and (P2) is given by: $A_M = \{a \mid |M| \in \bigcap_{n \in M} P'_n(a)'\}$. Let B be the bipartite graph having \mathcal{M} on one side and A on the other side and having an edge between a vertex $M \in \mathcal{M}$ and a vertex $a \in A$ if $a \in A_M$. We claim that I has a stable assignment satisfying (P1) and (P2) if and only if B has a matching that saturates $\mathcal{M} \cup A_{\neq \emptyset}$. Since deciding the existence of such a matching can be achieved in time $\mathcal{O}(\sqrt{|V(B)||E(B)|}) = \mathcal{O}(\sqrt{|N \cup A||N||A|})$ (see e.g. [13, Lemma 4]), establishing this claim is the last component required for the proof of the theorem.

Towards showing the forward direction, let π be a stable assignment for I satisfying (P1) and (P2). Then $O = \{(a, \phi^{-1}(a)) \mid a \in A\}$ is a matching in B that saturates $\mathcal{M} \cup A_{\neq \emptyset}$. Note that O saturates \mathcal{M} due to (P2), moreover, O saturates $A_{\neq \emptyset}$ since otherwise there would be an activity $a \in A_{\neq \emptyset}$ with $\pi^{-1}(a) = \emptyset$, which due to the definition of $A_{\neq \emptyset}$ and (P1) implies there is an agent n with $\pi(n) = a_\emptyset$ such that $1 \in P_n(a)$, contradicting our assumption that π is stable.

Towards showing the reverse direction, let O be a matching in B that saturates $\mathcal{M} \cup A$. Then the assignment π mapping all agents in M (for every $M \in \mathcal{M}$) to its partner in O and all other agents to a_\emptyset clearly already satisfies (P1) and (P2). It remains to show that it is also stable. Note that π is individually rational because of the construction of B . Moreover, assume for a contradiction that there is an agent $n \in N_\emptyset$ with $\pi(n) = a_\emptyset$ and an activity $a \in A$ such that $(a, |\pi^{-1}(a)| + 1) \in P_n$. If $|\pi^{-1}(a)| = 0$, then $a \in A_{\neq \emptyset}$ and hence $|\pi^{-1}(a)| > 0$ (because O saturates $A_{\neq \emptyset}$), a contradiction. If on the other hand $|\pi^{-1}(a)| \neq 0$, then $\{M, a\} \in O$ (for some $M \in \mathcal{M}$), but $(a, |\pi^{-1}(a)|) \notin P'_n$ and hence $\{M, a\} \notin E(B)$, also a contradiction. \blacktriangleleft

For future work, we believe that it would be interesting to see how the complexity map changes if one were to consider the number of activity types instead of the number of activities in our parameterizations.

References

- 1 Martin Aigner and Günter M. Ziegler. *Proofs from the Book (3. ed.)*. Springer, 2004.
- 2 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 3 Andreas Darmann. Group Activity Selection from Ordinal Preferences. In Toby Walsh, editor, *Algorithmic Decision Theory - 4th International Conference, ADT 2015, Lexington, KY, USA, September 27-30, 2015, Proceedings*, volume 9346 of *Lecture Notes in Computer Science*, pages 35–51. Springer, 2015.

- 4 Andreas Darmann, Janosch Döcker, Britta Dorn, Jérôme Lang, and Sebastian Schneckeburger. On Simplified Group Activity Selection. In Jörg Rothe, editor, *Algorithmic Decision Theory - 5th International Conference, ADT 2017, Luxembourg, Luxembourg, October 25-27, 2017, Proceedings*, volume 10576 of *Lecture Notes in Computer Science*, pages 255–269. Springer, 2017.
- 5 Andreas Darmann, Edith Elkind, Sascha Kurz, Jérôme Lang, Joachim Schauer, and Gerhard Woeginger. Group activity selection problem with approval preferences. *International J. of Game Theory*, pages 1–30, 2017.
- 6 Andreas Darmann, Edith Elkind, Sascha Kurz, Jérôme Lang, Joachim Schauer, and Gerhard J. Woeginger. Group Activity Selection Problem. In *Internet and Network Economics - 8th International Workshop, WINE 2012*, volume 7695 of *Lecture Notes in Computer Science*, pages 156–169. Springer, 2012.
- 7 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- 8 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer Verlag, 2013. doi:10.1007/978-1-4471-5559-1.
- 9 Eduard Eiben, Robert Galian, and Sebastian Ordyniak. A Structural Approach to Activity Selection. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, pages 203–209. ijcai.org, 2018.
- 10 Paul Erdős and Paul Turán. On a problem of Sidon in additive number theory, and on some related problems. *Journal of the London Mathematical Society*, 1(4):212–215, 1941.
- 11 Michael R. Fellows, Daniel Lokshantov, Neeldhara Misra, Frances A. Rosamond, and Saket Saurabh. Graph Layout Problems Parameterized by Vertex Cover. In *Algorithms and Computation, 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 15-17, 2008. Proceedings*, pages 294–305, 2008.
- 12 Robert Galian, Fabian Klute, and Sebastian Ordyniak. On Structural Parameterizations of the Bounded-Degree Vertex Deletion Problem. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 33:1–33:14, 2018.
- 13 Robert Galian, Sebastian Ordyniak, and Ramanujan Sridharan. On Structural Parameterizations of the Edge Disjoint Paths Problem. In Yoshio Okamoto and Takeshi Tokuyama, editors, *28th International Symposium on Algorithms and Computation, ISAAC 2017, December 9-12, 2017, Phuket, Thailand*, volume 92 of *LIPICs*, pages 36:1–36:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.
- 14 Sushmita Gupta, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi. Group Activity Selection on Graphs: Parameterized Analysis. In Vittorio Bilò and Michele Flammini, editors, *Algorithmic Game Theory - 10th International Symposium, SAGT 2017, L'Aquila, Italy, September 12-14, 2017, Proceedings*, volume 10504 of *Lecture Notes in Computer Science*, pages 106–118. Springer, 2017.
- 15 Ayumi Igarashi, Robert Brederick, and Edith Elkind. On Parameterized Complexity of Group Activity Selection Problems on Social Networks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017*, pages 1575–1577. International Foundation for Autonomous Agents and Multiagent Systems, 2017.
- 16 Ayumi Igarashi, Robert Brederick, and Edith Elkind. On Parameterized Complexity of Group Activity Selection Problems on Social Networks. *CoRR*, abs/1703.01121, 2017. arXiv:1703.01121.
- 17 Ayumi Igarashi, Robert Brederick, Dominik Peters, and Edith Elkind. Group Activity Selection on Social Networks. *CoRR*, abs/1712.02712, 2017. arXiv:1712.02712.
- 18 Ayumi Igarashi, Dominik Peters, and Edith Elkind. Group Activity Selection on Social Networks. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 565–571. AAAI Press, 2017.

48:16 Group Activity Selection with Few Agent Types

- 19 Hooyeon Lee and Virginia Vassilevska Williams. Parameterized Complexity of Group Activity Selection. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017*, pages 353–361. ACM, 2017.
- 20 Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. of Computer and System Sciences*, 67(4):757–771, 2003.


Optimal Sorting with Persistent Comparison Errors

Barbara Geissmann 

Department of Computer Science, ETH Zürich, Switzerland
barbara.geissmann@inf.ethz.ch

Stefano Leucci 

Department of Algorithms and Complexity, Max Planck Institute for Informatics, Germany*
<https://www.stefanoleucci.com>
stefano.leucci@mpi-inf.mpg.de

Chih-Hung Liu 

Department of Computer Science, ETH Zürich, Switzerland
chih-hung.liu@inf.ethz.ch

Paolo Penna 

Department of Computer Science, ETH Zürich, Switzerland
paolo.penna@inf.ethz.ch

Abstract

We consider the problem of sorting n elements in the case of *persistent* comparison errors. In this problem, each comparison between two elements can be wrong with some fixed (small) probability p , and *comparisons cannot be repeated* (Braverman and Mossel, SODA'08). Sorting perfectly in this model is impossible, and the objective is to minimize the *dislocation* of each element in the output sequence, that is, the difference between its true rank and its position. Existing lower bounds for this problem show that no algorithm can guarantee, with high probability, *maximum dislocation* and *total dislocation* better than $\Omega(\log n)$ and $\Omega(n)$, respectively, *regardless of its running time*.

In this paper, we present the first $O(n \log n)$ -time sorting algorithm that guarantees both $O(\log n)$ *maximum dislocation* and $O(n)$ *total dislocation* with high probability. This settles the time complexity of this problem and shows that comparison errors do not increase its computational difficulty: a sequence with the best possible dislocation can be obtained in $O(n \log n)$ time and, even without comparison errors, $\Omega(n \log n)$ time is necessary to guarantee such dislocation bounds.

In order to achieve this optimality result, we solve two sub-problems in the persistent error comparisons model, and the respective methods have their own merits for further application. One is how to locate a position in which to insert an element in an almost-sorted sequence having $O(\log n)$ maximum dislocation in such a way that the dislocation of the resulting sequence will still be $O(\log n)$. The other is how to simultaneously insert m elements into an almost sorted sequence of m different elements, such that the resulting sequence of $2m$ elements remains almost sorted.

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

Keywords and phrases approximate sorting, comparison errors, persistent errors

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.49

Related Version A full version of the paper is available at <https://arxiv.org/abs/1804.07575>.

Funding Research supported by SNF (project number 200021_165524).

Acknowledgements The authors wish to thank Peter Widmayer for many insightful discussions.

* Part of this work was completed while the author was affiliated with ETH Zürich.



1 Introduction

We study the problem of *sorting* n distinct elements under *persistent random comparison errors*, where each comparison is wrong with some fixed (small) probability p , and the errors are independent over all possible pairs of elements. There are two types of comparison errors, *persistent* and *non-persistent*. For non-persistent errors, it is possible to repeat the same comparison several times and each result is wrong with probability p independently of the others. In the 80s and 90s, non-persistent errors have received considerable attention, and it has been shown that the perfectly sorted sequence can be computed in $O(n \log n)$ time with high probability. For persistent errors the repetition of a single comparison always yields the same outcome and this makes *impossible* to consistently recover the *perfectly sorted* sequence, as we explain below. The goal is therefore that of computing an *almost sorted* sequence. This seems a challenging task as all known algorithms have rather high running time and only recently a sub-quadratic running time has been achieved (see below for details). In particular, whether an optimal $O(n \log n)$ running time is sufficient for sorting with *persistent* comparison errors is a fundamental open question.

The above persistent-errors model is a well-studied theoretical abstraction of the errors that arise in hardware architectures. Here, avoiding these errors requires involved fault-tolerant mechanisms that reduce performances and increase manufacturing costs and energy consumption. Recently, a contrasting trend of simplifying hardware architectures has emerged: errors are traded for cheaper manufacturing costs, lower energy consumption, or better performances. The study of sorting algorithms with persistent comparison errors has been also motivated in [5, 13] by experts comparing items according to their importance, by ranking in sports where comparisons correspond to matches between teams, and –more generally– by situations where one wants to aggregate noisy comparisons into a global ranking and repeating a comparison is impossible or too expensive.

A common way to measure the quality of an output sequence in terms of sortedness, is to consider the *dislocation* of an element, which is the difference between its position in the output and its position in the correctly sorted sequence. In particular, a reasonable measure is the *maximum dislocation* of any element in the sequence or the *total dislocation* of the sequence, i.e., the sum of the dislocations of all n elements.

To see why sorting with persistent errors is much more difficult than the case in which comparisons can be repeated, note that in the latter case there is a trivial $O(n \log^2 n)$ time solution to sort perfectly with high probability (simply repeat each comparison $O(\log n)$ times and take the majority of the results). Instead, in the model with persistent errors, it is impossible to sort perfectly as, for any constant p , no algorithm can achieve a maximum dislocation that is smaller than $\Omega(\log n)$ w.h.p., or total dislocation smaller than $\Omega(n)$ in expectation [10]. This problem has been extensively studied in the literature, and several algorithms have been devised with the goal of sorting *quickly* with small dislocation (see Table 1). Unfortunately, even though all the algorithms achieve the best possible maximum dislocation of $\Theta(\log n)$, they use a truly superlinear number of comparisons (specifically, $\Omega(n^c)$ with $c \geq 1.5$), and/or require significant amount of time (namely, $O(n^{3+c_p})$ where c_p is a big constant that depends on p). This naturally suggests the following question:

What is the time complexity of sorting optimally with persistent errors?

In this work, we answer this basic question by showing the following result:

*There exists an algorithm with **optimal running time** $O(n \log n)$ which achieves simultaneously **optimal maximum dislocation** $O(\log n)$ and **optimal total dislocation** $O(n)$, both with high probability.*

■ **Table 1** The existing approximate sorting algorithms and our result. The constant c_p in the exponent of the running time of [5] depends on the error probability p and it is typically quite large. We write $\Omega(f(n))$ w.h.p. (resp. exp.) to mean that no algorithm can achieve dislocation $o(f(n))$ with high probability (resp. in expectation).

Upper bounds

Running Time	Max Dislocation	Tot Dislocation	Reference
$O(n^{3+c_p})$	$O(\log n)$ w.h.p.	$O(n)$ w.h.p.	[5]
$O(n^2)$	$O(\log n)$ w.h.p.	$O(n \log n)$ w.h.p.	[13]
$O(n^2)$	$O(\log n)$ w.h.p.	$O(n)$ expected	[10]
$\tilde{O}(n^{3/2})$	$O(\log n)$ w.h.p.	$O(n)$ expected	[11]
$O(n \log n)$	$O(\log n)$ w.h.p.	$O(n)$ w.h.p.	this work

Lower bounds

Any	$\Omega(\log n)$ w.h.p.	$\Omega(n)$ expected	[10]
-----	-------------------------	----------------------	------

The dislocation guarantees of our algorithm are optimal, due to the lower bound of [10], while the existence of an algorithm achieving a maximum dislocation of $d = O(\log n)$ in time $T(n) = o(n \log n)$ would immediately imply the existence of an algorithm that sorts n elements in $T(n) + O(n \log \log n) = o(n \log n)$ time, even in the absence of comparison errors, thus contradicting the classical $\Omega(n \log n)$ lower bound for comparison-based algorithms.¹ Along the way to our result, we consider the problem of *searching with persistent errors*, defined as follows:

We are given an approximately sorted sequence S , and an additional element $x \notin S$. The goal is to compute, under persistent comparison errors, an approximate rank (position) of x which differs from the true rank of x in S by a small additive error.

For this problem, we show an algorithm that requires $O(\log n)$ time to compute, w.h.p., an approximate rank that differs from the true rank of x by at most $O(\max\{d, \log n\})$, where d is the maximum dislocation of S . For $d = \Omega(\log n)$ this allows to insert x into S without any asymptotic increase of the maximum (and total) dislocation in the resulting sequence. Notice that, if d is also in $O(n^{1-\epsilon})$ for any constant $\epsilon > 0$, this is essentially the best we can hope for, as an easy decision-tree lower bound shows that any algorithm must require $\Omega(\log n)$ time. Finally, we remark that [13] considered the variant in which the original sequence is *sorted*, and the algorithm must compute the correct rank. For this problem, they present an algorithm that runs in $O(\log n \cdot \log \log n)$ time and succeeds with probability $1 - f(p)$, with $f(p)$ vanishing as p goes to 0. As by-product of our result, we can obtain the optimal $O(\log n)$ running time with essentially the same success probability. Similarly to other prior related works, all our results apply when p is below a sufficiently small constant, e.g., $p < 1/20$ in [13]. For technical simplicity, throughout this work we assume $p < 1/32$, though the results hold for $p < 1/16$ as in [10].²

¹ Indeed, the smallest d elements of a sequence S having dislocation $d = 2^{o(\log n)}$ can be found in time $O(d \log d)$ using any $O(n \log n)$ -time sorting algorithm on the first $2d$ elements of S . Removing those elements and repeating the above procedure $O(\frac{n}{d})$ times, would allow to sort in $T(n) + O(\frac{n}{d} \cdot d \log d) = o(n \log n)$ time.

² Except for the derandomization technique of Section 5, all our results also hold for the case in which each comparison is wrong with an adversarially chosen and unknown probability in $[0, p]$.

1.1 Main Intuition and Techniques

Approximate Sorting

In order to convey the main intuitions behind our $O(n \log n)$ -time optimal-dislocation approximate sorting algorithm, we consider the following ideal scenario: we already have a perfectly sorted sequence A containing a random half of the elements in our input sequence S and we, somehow, also know the position in which each element $x \in S \setminus A$ should be inserted into A so that the resulting sequence is also sorted (i.e., the *rank* of x in A). If these positions alternate with the elements of A , then, to obtain a sorted version of S , it suffices to *merge* S and $S \setminus A$, i.e., to simultaneously insert all the elements of $S \setminus A$ into their respective positions of A . Unfortunately, we are far from this ideal scenario for several reasons: first of all, multiple elements in $S \setminus A$, say δ of them, might have the same rank in A . Since we do not know the order in which those elements should appear, this will already increase the dislocation of the merged sequence to $\Omega(\delta)$. Moreover, due to the lower bound of [10], we are not actually able to obtain a perfectly sorted version of A and we are forced to work with a permutation of A having dislocation $d = \Omega(\log n)$, implying that the natural bound on the resulting dislocation can be as large as $d \cdot \delta$. This is bad news, as one can show that $\delta = \Omega(\log n)$. However, it turns out that the number of elements in $S \setminus A$ whose positions lie in a $O(\log n)$ -wide interval of A is still $O(\log n)$, w.h.p., implying that the final dislocation of A is just $O(\log n)$.

But how do we obtain the approximately sorted sequence A in the first place? We could just recursively apply the above strategy on the (unsorted) elements of A , except that this would cause a blow-up in the resulting dislocation due to the constant hidden by the big-O notation. We therefore interleave merge steps with invocations of (a modified version of) the sorting algorithm of [10], which essentially reduces the dislocation by a constant factor, so that the increase in the worst-case dislocation will be only an *additive* constant per recursive step.

An additional complication is due to the fact that we are not able to compute the exact ranks in A of the elements in $S \setminus A$. We therefore have to deal, once again, with approximations that are computed using the other main contribution of this paper: *noisy binary search trees*, whose key ideas are described in the following.

Noisy Binary Search

As a key ingredient of our approximate sorting algorithm, we need to *merge* an almost-sorted sequence with a set of elements, without any substantial increase in the final maximum dislocation. More precisely, given a sequence S with maximum dislocation d and an element $x \notin S$, we want to compute an *approximate rank* of x in S , i.e., a position that differs by $O(\max\{d, \log n\})$ from the position that x would occupy if the elements $S \cup \{x\}$ were perfectly sorted. This same problem has been solved optimally in $O(\log n)$ time in the easier case in which errors are *not persistent* and S is already sorted [9]. The idea of [9] is to locate the correct position of x using a binary decision tree: ideally each vertex v of the tree *tests* whether x appears to belong to a certain *interval* of S and, depending on the result, one of the children of v is considered next. Since these intervals become narrower as we move from the root towards the leaves (that are in a one-to-one correspondence with positions of S) we eventually discover the correct rank of x in S . In order to cope with failures, this process is allowed to *backtrack* when inconsistent comparisons are observed, thus repeating some of the comparisons involving ancestors of v . Moreover, to obtain the correct result with high probability, a logarithmic number of consistent comparisons with a leaf are needed before the algorithm terminates.

Notice how the above process relies on the fact that it is possible to gather more information on the true relative position of x by repeating a comparison multiple times (in fact, it is trivial to design a simple $O(\log^2 n)$ -time algorithm in this error model). Unfortunately, this is no longer the case when errors are *persistent*. To overcome this problem we design a *noisy binary search tree* in which testing whether x belongs to the interval associated with a vertex v also causes the interval itself to *grow* thus ensuring that, in future tests involving v , x will always be compared with different elements. This, however, is a source of other difficulties: first, the intervals of the descendants of v also need to be suitably updated to account for the new elements in v 's interval. Moreover, since intervals are now *dynamic*, it is possible for multiple tests on the same vertex to report different results even when no comparison errors occur: this is because an interval that did not initially contain x might eventually grow into one that does. Finally, since growing intervals need to overlap, one also has to be careful in avoiding repeated comparisons arising from *unrelated* vertices (i.e., vertices that are not in ancestor–descendant relation in the tree). We overcome these problems by using two search trees that initially comprise of disjoint intervals and ensure that all the vertices exhibiting the problematic behaviours discussed above will be confined to only one of the two trees: in some sense, we guarantee that one of two search trees will behave similarly to the one of [9], where leaves now represent groups of $O(\log n)$ positions in S .

1.2 Related work

Sorting with *persistent errors* has been studied in several works, starting from [5] who presented the first algorithm achieving optimal dislocation (matching lower bounds appeared only recently in [10]) by finding a maximum-likelihood permutation of the input elements given the observed errors. The algorithm in [5] uses only $O(n \log n)$ comparisons and is able to handle any constant comparison error probability $p \in (0, \frac{1}{2})$ (later improved to $p \leq \frac{1}{2} - \Omega(\frac{\log \log n}{\log n})^{\frac{1}{6}}$ in [17]), but unfortunately its running time $O(n^{3+c_p})$ is quite large. For example, if we require the algorithm to succeed with a probability of $1 - 1/n$, the analysis in [5] yields $c_p = \frac{110525}{(1/2-p)^4}$. On the contrary, all subsequent faster algorithms [10,11,13] – see Table 1 – use a number of comparisons which is asymptotically equal to their respective running time and work for a smaller range of values of p (i.e., $p \leq \frac{1}{20}$ in [13] and $p < \frac{1}{16}$ in [10,11]).

Other works considered error models in which repeating comparisons is possible, although expensive. For example, [4] studied algorithms which use a *bounded number of rounds* for some “easier” versions of sorting (e.g., distinguishing the top k elements from the others). Note that each round consists of a set of comparison operations, where it is possible to compare the same pair of elements several times using independent comparisons like in the non-persistent model; Also, the comparisons made in each round are decided a priori, i.e., they do not depend on the results of the comparisons in this round. In each round, a fresh set of comparison results is generated, and each round consists of $\delta \cdot n$ comparisons. They evaluate the algorithm’s performance by estimating the number of “misclassified” elements and also consider a variant in which errors now correspond to missing comparison results.

In general, sorting in presence of errors seems to be computationally more difficult than the error-free counterpart. For instance, [1] provides algorithms using *subquadratic* time (and number of comparisons) when errors occur only between elements whose difference is at most some fixed threshold. Also, [8] gives a *subquadratic* time algorithm when the number k of errors is known in advance.

As mentioned above, an easier error model is the one with *non-persistent* errors, meaning that the same comparison can be *repeated* and the errors are independent, and happen with some probability $p < 1/2$. In this model it is possible to sort n elements in time $O(n \log(n/q))$, where $1 - q$ is the success probability of the algorithm [9] (see also [2,12] for the analysis of the classical Quicksort and recursive Mergesort algorithms in this error model).

More generally, computing with errors is often considered in the framework of a two-person game called *Rényi-Ulam Game*. In this game a questioner tries to identify an unknown object x from a universe U by asking yes-or-no questions to a responder, but some of the answers might be wrong. The case in which $U = \{1, \dots, n\}$, the questions are of the form “is $s > x$?”, and each answer is independently incorrect with probability $p < \frac{1}{2}$ has been considered by [9], where the authors provide a binary search algorithm that succeeds with probability $1 - \delta$ and requires $O(\log \frac{n}{\delta})$ worst-case time. In [3], the authors then showed how to find s using an optimal amount of queries up to additive polylogarithmic terms. The variant in which responder is allowed to adversarially lie up to k times has been proposed by Rényi [15] and Ulam [18], which has then been solved by Rivest et al. [16] using only $\log n + k \log \log n + O(k \log k)$ question, which is tight. Among other results, near-optimal strategies for the distributional version of the game have been devised in [7]. For more related results on the topic, we refer the interested reader to [14] for a survey and to [6] for a monograph.

1.3 Paper Organization

The rest of this work is organized as follows: in Section 2 we give some preliminary definitions; then, in Section 3, we present our noisy binary search algorithm, which will be used in Section 4 to design an optimal randomized sorting algorithm. Finally, in Section 5, we briefly argue on how our sorting algorithm can be adapted so that it does not require any external source of randomness. Due to space limitations, this manuscript only includes the core parts of the analysis of our sorting algorithm. We refer the reader to the full version of the paper for the formal analysis of other claims of Section 4, and of the results in the remaining sections. Moreover, Section 4 makes use of an improved analysis of the sorting algorithm of [10] which can also be found in the full version of the paper.

2 Preliminaries

According to our error model, elements possess a true total linear order, however this order can only be observed through noisy comparisons. In the following, given two distinct elements x and y , we will write $x \prec y$ (resp. $x \succ y$) to mean that x is smaller (resp. larger) than y according to the true order, and $x < y$ (resp. $x > y$) to mean that x appears to be smaller (resp. larger) than y according to the observed comparison result.

Given a sequence or a set of elements A and an element x (not necessarily in A), we define $\text{rank}(x, A) = |\{y \in A : y \prec x\}|$ as the *true rank* of element x in A (notice that ranks start from 0). Moreover, if A is a sequence and $x \in A$, we denote by $\text{pos}(x, A) \in [0, |A| - 1]$ the *position* of x in A (notice that positions are also indexed from 0), so that the *dislocation* of x in A is $\text{disl}(x, A) = |\text{pos}(x, A) - \text{rank}(x, A)|$, and the *maximum dislocation* of the sequence A is $\text{disl}(A) = \max_{x \in A} \text{disl}(x, A)$.

For $z \in \mathbb{R}^+$, $\ln z$ and $\log z$ refer to the natural and the binary logarithm of z , respectively.

3 Noisy Binary Search

Given a sequence $S = \langle s_0, \dots, s_{n-1} \rangle$ of n elements with maximum dislocation $d \geq \log n$, and an additional element x not in S , we want to compute in time $O(\log n)$ an *approximate rank* of x in S , that is, a position where to insert x in S while preserving a $O(d)$ upper bound on dislocation of the resulting sequence. More precisely, we want to compute index r_x such that $|r_x - \text{rank}(x, S)| = O(d)$, in presence of persistent comparison errors: Errors between x and

the elements in S happen independently with probability p , and whether the comparison between x and an element $y \in S$ is correct or erroneous does not depend on the position of y in S , nor on the actual permutation of the sorted elements induced by their order in S (i.e., we are not allowed to pick the order of the elements in S as a function of the comparison errors involving x). We do not impose any restriction on the errors of comparisons that do not involve x .

In the following, we will show an algorithm that computes such a rank r_x in time $O(\log n)$. This immediately implies that $O(\log n)$ time also suffices to insert x into S so that the resulting sequence $\langle s_0, \dots, s_{r_x-1}, x, s_{r_x}, s_{n-1} \rangle$ still has maximum dislocation $O(d)$.

► **Remark 1.** The $O(\log n)$ running time is asymptotically optimal for all $d = n^{1-\epsilon}$, for constant $\epsilon < 1$, since a $\Omega(\log n - \log d) = \Omega(\log n)$ decision-tree lower bound holds even in absence of comparison errors.

In the following, for the sake of simplicity, we let $c = 10^3$ and we assume that $n = 2cd \cdot 2^h - 1$ for some non-negative integer h . Moreover, we focus on $p \leq \frac{1}{32}$ even though this restriction can be easily removed to handle all constant $p < \frac{1}{2}$, as we argue at the end of the section.

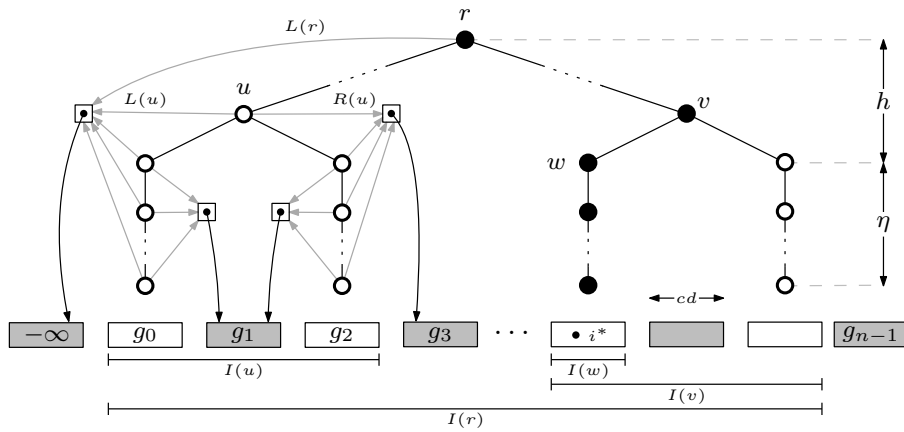
We consider the set $\{0, \dots, n\}$ of the possible ranks of x in S and we subdivide them into $2 \cdot 2^h$ ordered groups g_0, g_1, \dots each containing cd contiguous positions, namely, group g_i contains positions $cid, \dots, c(i+1)d - 1$. Then, we further partition these $2 \cdot 2^h$ groups into two ordered sets G_0 and G_1 , where G_0 contains the groups g_i with even i ($i \equiv 0 \pmod{2}$) and G_1 the groups g_i with odd i ($i \equiv 1 \pmod{2}$). Notice that $|G_0| = |G_1| = 2^h$. In the next section, for each G_j , we shall define a *noisy binary search tree* T_j , which will be the main ingredient of our algorithm.

3.1 Constructing T_0 and T_1

Let us consider a fixed $j \in \{0, 1\}$ and define $\eta = 2 \lceil \log n \rceil$. The tree T_j comprises of a binary tree of height $h + \eta$ in which the first $h + 1$ levels (i.e., those containing vertices at depths 0 to h) are complete and the last η levels consists of 2^h paths of η vertices, each emanating from a distinct vertex on the $(h + 1)$ -th level. We index the leaves of the resulting tree from 0 to $2^h - 1$, we use $h(v)$ to denote the depth of vertex v in T_j , and we refer to the vertices v at depth $h(v) \geq h$ as *path-vertices*. Each vertex v of the tree is associated with one *interval* $I(v)$, i.e., as a set of contiguous positions, as follows: for a leaf v having index i , $I(v)$ consists of the positions in g_{2^i+j} ; for a non-leaf path-vertex v having u as its only child, we set $I(v) = I(u)$; finally, for an internal vertex v having u and w as its left and right children, respectively, we define $I(v)$ as the interval containing all the positions between $\min I(u)$ and $\max I(w)$, endpoints included.

Moreover, each vertex v of the tree has a reference to two *shared pointers* $L(v)$ and $R(v)$ to positions in $\mathbb{Z} \setminus \bigcup_{g_i \in G_j} g_i$. Intuitively, $L(v)$ (resp. $R(v)$) will always point to positions of S occupied by elements that are *smaller* (resp. *larger*) than all the elements s_i with $i \in I(v)$. For each leaf v , let $L(v)$ initially point to $\min I(v) - d - 1$ and $R(v)$ initially point to $\max I(v) + d$. A non-leaf path-vertex v shares both its pointers with the corresponding pointers of its only child, while a non-path vertex v shares its left pointer $L(v)$ with the left pointer of its left child, and its right pointer $R(v)$ with the right pointer of its right child. See Figure 1 for an example.

Notice that we sometimes allow $L(v)$ to point to negative positions and $R(v)$ to point to positions that are larger than $n - 1$. In the following we consider all the elements s_i with $i < 0$ (resp. $i \geq n$) to be copies a special $-\infty$ (resp. $+\infty$) element such that $-\infty \prec x$ and $-\infty < x$ in every observed comparison (resp. $+\infty \succ x$ and $+\infty > x$).



■ **Figure 1** An example of the noisy tree T_0 . On the left side the shared pointers $L(\cdot)$ and $R(\cdot)$ are shown. Notice how $L(r)$ (and, in general, all the $L(\cdot)$ pointers on the leftmost side of the tree) points to the special $-\infty$ element. Good (resp. bad) vertices are shown in black (resp. white). Notice that, since $i^* \in I(w)$, $T^* = T_0$ and all the depicted vertices are either good or bad.

3.2 Walking on T_j

The algorithm will perform a discrete-time random walk on each T_j . Before describing such a walk in more detail, it is useful to define the following operation:

► **Definition 2** (test operation). A test of an element x with a vertex v is performed by (i) comparing x with the elements $s_{L(v)}$ and $s_{R(v)}$, (ii) decrementing $L(v)$ by 1 and, (iii) incrementing $R(v)$ by 1. The tests succeeds if the observed comparison results are $x > s_{L(v)}$ and $x < s_{R(v)}$, otherwise the test fails.

The walk on T_j proceeds as follows. At time 0, i.e., before the first step, the *current* vertex v coincides with the root r of T_j . Then, at each time step, we *walk* from the current vertex v to the next vertex as follows:

1. We test x with all the children of v and, if *exactly one* of these tests succeeds, we *walk to the corresponding child*.
2. Otherwise, if *all the tests fail*, we *walk to the parent* of v , if it exists.

In the remaining cases we “walk” from v to itself. We also define $\tau = 240 \lceil \log n \rceil$ and we stop the walk as soon as one of the following two conditions is met:

Success: The current vertex v is a leaf of T_j . In this case we say that the walk *returns* v ;

Timeout: The τ -th time step is completed and the success condition is not met.

It turns out that at least one of the walks on T_0 and T_1 will succeed w.h.p., while the other can either succeed or timeout. If any of the walks succeeds and returns v , we output any position in the interval $I(v)$. Otherwise, we return an arbitrary position. We are then to prove the following result, whose analysis can be found in the full version of the paper:

► **Theorem 3.** Let S be a sequence of n elements having maximum dislocation at most $d \geq \log n$ and let $x \notin S$. Under our error model, an index $r_x \in [\text{rank}(x, S) - \alpha d, \text{rank}(x, S) + \alpha d]$ can be found in $O(\log n)$ time with probability at least $1 - O(n^{-6})$, where $\alpha > 1$ is an absolute constant.

To conclude this section, we remark that our assumption that $p \leq \frac{1}{32}$ can be easily relaxed to handle any constant error probability $p < \frac{1}{2}$. This can be done by modifying the test operation so that, when x is tested with a vertex v , the majority result of the

comparisons between x and the set $\{s_{L(v)}, s_{L(v)-1}, \dots, s_{L(v)-k+1}\}$ (resp. x and the set $\{s_{R(v)}, s_{R(v)+1}, \dots, s_{R(v)+k-1}\}$) of η elements is considered, where k is a constant that only depends on p . Consistently, the pointers $L(v)$ and $R(v)$ are shifted by k positions, and the group size is increased to $k \cdot c$. Notice how our description for $p \leq \frac{1}{32}$ corresponds exactly to the case $k = 1$. The only difference in the statement Theorem 3 is that α is no longer an absolute constant, but rather, it depends (only) on the value of p .

4 Optimal Sorting Algorithm

4.1 The algorithm

Here we present an optimal sorting algorithm that, given a sequence S of n elements, computes, in $O(n \log n)$ worst-case time, a permutation of S having maximum dislocation $O(\log n)$ and total dislocation $O(n)$, w.h.p. In order to avoid being distracted by roundings, we assume that n is a power of two.³ Our algorithm will make use of the noisy binary search of Section 3 and of the `WindowSort` algorithm [10]. For our purposes, we need the following *stronger* version of the original result in [10], in which the bound on the total dislocation was only given in expectation:

► **Theorem 4.** *Consider a set of n elements that are subject to random persistent comparison errors. For any dislocation d , and for any (adversarially chosen) permutation S of these elements such that $\text{disl}(S) \leq d$, `WindowSort`(S, d) requires $O(nd)$ worst-case time to compute, with probability at least $1 - \frac{1}{n^4}$, a permutation of S having maximum dislocation at most $c_p \cdot \min\{d, \log n\}$ and total dislocation at most $c_p \cdot n$, where c_p is a constant depending only on the error probability $p < \frac{1}{32}$.*

We give a brief description of `WindowSort` and prove the above theorem in Section 5 of the full version of the paper. Notice that `WindowSort` also works in a stronger error model in which S can be chosen adversarially after the comparison errors between all pairs of elements have been randomly fixed, as long as its maximum dislocation is at most d . In the remaining of this section, we assume $p \leq 1/32$ in order to be consistent with Section 3, though both the above theorem and the algorithm we are going to present will only require $p < 1/16$.⁴ Using the noisy binary search in Section 3, we now define an operation that allows us to add a linear number of elements to an almost-sorted sequence without any asymptotic increase in the resulting dislocation, as we will formally prove in the sequel. More precisely, if A and B are two disjoint subsets of S , we denote by `Merge`(A, B) the sequence obtained as follows:

- For each $x \in B$, compute an index r_x such that $|\text{rank}(s, A) - r_x| \leq \alpha d$. This can be done using the noisy binary search of Section 3, which succeeds with probability at least $1 - \frac{1}{|A|^6}$.
- Insert *simultaneously* all the elements $x \in B$ into A in their computed positions r_x , breaking ties arbitrarily. Return the resulting sequence.

Our sorting algorithm, that we call `RiffleSort` (see the pseudocode in Algorithm 1), works as follows. For $k = \frac{\log n}{2}$, we first partition S into $k + 1$ subsets T_0, T_1, \dots, T_k : Each T_i , with $1 \leq i \leq k$, contains $2^{i-1} \sqrt{n}$ elements chosen uniformly at random from $S \setminus \{T_{i+1}, T_{i+2}, \dots, T_k\}$, and $T_0 = S \setminus \{T_1, T_2, \dots, T_k\}$ contains the remaining $n - \sqrt{n} \sum_{i=1}^k 2^{i-1} = \sqrt{n}$ elements. As its

³ This assumption can be easily removed by adding dummy $+\infty$ elements to S . Since `WindowSort`, the noisy binary search of Section 3, and ultimately our algorithm will also work when p is an upper bound on the error probability, it is not necessary to simulate errors when comparisons involving dummy elements are performed.

⁴ In fact, Theorem 4 is the only reason preventing our novel sorting algorithm to work for any constant $p \in [0, \frac{1}{2})$.

■ **Algorithm 1** RiffleSort(S).

```

1  $T_0, T_1, \dots, T_k \leftarrow$  partition of  $S$  computed as explained in Section 4.1;
2  $S_0 \leftarrow$  WindowSort( $T_0, \sqrt{n}$ );
3 foreach  $i = 1, \dots, k = \frac{\log n}{2}$  do
4    $S_i \leftarrow$  Merge( $S_{i-1}, T_i$ );
5    $S_i \leftarrow$  WindowSort( $S_i, \gamma \cdot c_p \cdot \log n$ );
6 return  $S_k$ ;

```

first step, RiffleSort will approximately sort T_0 using WindowSort, and then it will alternate merge operations with calls to WindowSort. On one hand the merge operations allow us to iteratively grow the set of approximately sorted elements to ultimately include all the elements in S but, on the other hand, each operation also increases the dislocation by a constant factor. This is a problem since the rate at which the dislocation increases is faster than the rate at which new elements are inserted. The role of the sorting operations is exactly to circumvent this issue: each WindowSort call has the effect of locally rearranging the elements, so that all newly inserted elements are now closer to their intended positions, causing (an upper bound to) the resulting maximum dislocation to increase by only an additive constant. The corresponding pseudocode is shown in Algorithm 1, in which $\gamma \geq \max\{202\alpha, 909\}$ is an absolute constant (recall that α is the constant from Theorem 3).

4.2 Analysis

► **Lemma 5.** *The worst-case running time of Algorithm 1 is $O(n \log n)$.*

Proof. Clearly the random partition T_0, \dots, T_k can be computed in time $O(n \log n)$,⁵ and the first call to WindowSort requires time $O(|T_0| \cdot \sqrt{n}) = O(n)$ (see Theorem 4). We can therefore restrict our attention to the generic i -th iteration of the for loop. The call to Merge(S_{i-1}, T_i) can be performed in $O(|S_i| \log n)$ time since, for each $x \in T_i$, the required approximation of $\text{rank}(x, S_{i-1})$ can be computed in time $O(\log |S_{i-1}|)$ and $|T_i| = |S_{i-1}| \leq n$, while inserting the elements of T_i in their computed ranks requires linear time in $|S_{i-1}| + |T_i| = |S_i|$. The subsequent execution of WindowSort with $d = O(\log n)$ requires time $O(|S_i| \log n)$, where the hidden constant does not depend on i . Therefore, for a suitable constant c , the time spent in the i -th iteration is $c|S_i| \log n$ and total running time of Algorithm 1 can be upper bounded by:

$$c \sum_{i=1}^k |S_i| \log n = c\sqrt{n} \log n \cdot \sum_{i=1}^k 2^i < 2^{k+1} c\sqrt{n} \log n = 2cn \log n. \quad \blacktriangleleft$$

The following lemma, that concerns a thought experiment involving urns and randomly drawn balls, will be useful to upper bound the dislocation of the sequences returned by the Merge operations. Since it can be proved using arguments that do not depend on the details of RiffleSort, we omit its proof, which can be found in the full version of the paper.

⁵ The exact complexity depends on whether we can sample u.a.r. an integer from a range in $O(1)$ time. If this is not the case, then integers can be generated bit-by-bit using rejection, and the total number of required random bits will be $O(n)$ with probability at least $1 - n^{-2}$, as shown in the full version of this paper. To maintain a worst-case upper bound on the running time also in the unlikely event that $\Theta(n \log n)$ bits do not suffice, we can stop the algorithm and return any arbitrary permutation of S .

► **Lemma 6.** *Consider an urn containing $N = 2M$ balls, M of which are white, while the remaining M are black. Balls are iteratively drawn from the urn without replacement until the urn is empty. If N is sufficiently large and $9 \log N \leq k \leq \frac{N}{16}$ holds, the probability that any contiguous subsequence of at most $100k$ drawn balls contains k or fewer white balls is at most N^{-6} .*

We can now show that, if A and B contain randomly selected elements, the dislocation of $\text{Merge}(A, B)$ is likely to be at most a constant factor larger than the dislocation of A :

► **Lemma 7.** *Let A be a sequence containing m randomly chosen elements from S and having maximum dislocation at most d , with $\log n \leq d = o(m)$. Let B be a set of m randomly chosen elements from $S \setminus A$. Then, for a suitable constant γ , and for large enough values of m , $\text{merge}(A, B)$ has maximum dislocation at most γd with probability at least $1 - m^{-4}$.*

Proof. Let $\beta = \max\{\alpha, 9/2\}$, $S' = \text{Merge}(A, B)$, and $S^* = \langle s_0^*, s_1^*, \dots, s_{2m-1}^* \rangle$ be the sequence obtained by sorting S' according to the true order of its elements. Assume that:

- all the approximate ranks r_x , for $x \in B$, are such that $|r_x - \text{rank}(x, A)| \leq \beta d$; and
- all the contiguous subsequences of S^* containing no more than $2\beta d + 2$ elements in A have length at most $200\beta d + 200$.

We will show in the sequel that the above assumptions are likely to hold.

Pick any element $x \in S'$. We now show that our assumptions imply that the dislocation of x in S' is at most $201d$. An element $y \in B$ can affect the final dislocation of x in S' only if one of the following two (mutually exclusive) conditions holds: (i) $y \prec x$ and $r_y \geq r_x$, or (ii) $y \succ x$ and $r_y \leq r_x$. All the remaining elements in B will be placed in the correct relative order w.r.t. x in S' , and hence they do not affect the final dislocation of x . If (i) holds, we have:

$$r_x - \beta d \leq r_y - \beta d \leq \text{rank}(y, A) \leq \text{rank}(x, A) \leq r_x + \beta d,$$

while, if (ii) holds, we have:

$$r_x - \beta d \leq \text{rank}(x, A) \leq \text{rank}(y, A) \leq r_y + \beta d \leq r_x + \beta d,$$

and hence, all the elements $y \in B$ that can affect the dislocation of x in S' are contained in the set $Y = \{y \in B : r_x - \beta d \leq \text{rank}(y, A) \leq r_x + \beta d\}$.

We now upper bound the cardinality of Y . Let y^- be the $(r_x - \beta d - 1)$ -th element of A ; if no such element exists, then let $y^- = s_0^*$. Similarly, let y^+ be the $(r_x + \beta d)$ -th element of A ; if no such element exists, then let $y^+ = s_{2m-1}^*$. Due to our choice of y^- and y^+ we have that $\forall y \in Y, y^- \preceq y \preceq y^+$, implying that all the elements in Y appear in the contiguous subsequence \bar{S} of S^* having y^- and y^+ as its endpoints. Since no more than $2\beta d + 2$ elements of A belong to \bar{S} , our assumption guarantees that \bar{S} contains at most $200\beta d + 200$ elements. This implies that the dislocation of x in S' is at most $\beta d + |Y| \leq \beta d + |\bar{S}| \leq 201\beta d + 200 \leq \gamma d$, where the last inequality holds for large enough n once we choose $\gamma = 202\beta$.

To conclude the proof we need to show that our assumptions hold with probability at least $1 - |S'|^{-4}$. Regarding the first assumption, for $x \in B$, a noisy binary search returns a rank r_x such that $|r_x - \text{rank}(x, A)| \leq \alpha d \leq \beta d$ with probability at least $1 - O(\frac{1}{m^6})$. Therefore the probability that the assumption holds is at least $1 - O(\frac{1}{m^5})$.

Regarding our second assumption, notice that, since the elements in A and B are randomly selected from S , we can relate their distribution in S^* with the distribution of the drawn balls in the urn experiment of Lemma 6: the urn contains $N = 2m$ balls each corresponding to an element in $A \cup B$, a ball is white if it corresponds to one of the $M = m$ elements of A ,

49:12 Optimal Sorting with Persistent Comparison Errors

while a black ball corresponds one of the $M = m$ elements of B . If the assumption does not hold, then there exists a contiguous subsequence of S^* of at least $200\beta d + 200$ elements that contains at most $2\beta d + 2$ elements from A . By Lemma 6 with $k = 2\beta d + 2$, this happens with probability at most $(2m)^{-6}$ (for sufficiently large values of n). The claim follows by using the union bound. \blacktriangleleft

We can now use Lemma 7 and Theorem 4 together to derive an upper bound to the final dislocation of the sequence returned by Algorithm 1.

► **Lemma 8.** *The sequence returned by Algorithm 1 has maximum dislocation $O(\log n)$ and total dislocation $O(n)$ with probability at least $1 - \frac{1}{n\sqrt{n}}$.*

Proof. For $i = 1, \dots, k$, we say that the i -th iteration of Algorithm 1 is *good* if the sequence S_i computed at its end has both (i) maximum dislocation at most $c_p \log n$, and (ii) total dislocation at most $c_p |S_i|$. As a corner case, we say that iteration 0 is good if S_0 also satisfies conditions (i) and (ii) above, which happens with probability at least $1 - \frac{1}{|S_0|^4} \geq 1 - \frac{1}{n^2}$ as shown by Theorem 4.

We now focus on a generic iteration $i \geq 1$ and show that, assuming that iteration $i - 1$ is good, iteration i is also good with probability at least $1 - \frac{1}{n^2}$. Since iteration $i - 1$ was good, the sequence S_{i-1} has maximum dislocation $c_p \log n$ and hence, by Lemma 7, the sequence resulting from call to `Merge`(S_{i-1}, T_i) returns a sequence with dislocation at most $\gamma c_p \log n$ with probability at least $1 - \frac{1}{|T_i|^4} \geq 1 - \frac{1}{n^2}$. If this is indeed the case, we have that the sequence S_i returned by the subsequent call to `WindowSort` satisfies (i) and (ii) with probability at least $1 - \frac{1}{|S_{i+1}|^4} \geq 1 - \frac{1}{n^2}$ (see Theorem 4). The claim follows by using the union bound on the $k = O(\log n)$ iterations, and by noticing that the returned sequence is exactly S_k . \blacktriangleleft

We can therefore state the following result, which follows directly from Lemma 8 and Lemma 5:

► **Theorem 9.** *Consider a set of n elements that are subject to random persistent comparison errors. For any (adversarially chosen) input permutation of these elements, `RiffleSort` is a randomized algorithm that returns, in $O(n \log n)$ worst-case time, a sequence having maximum (resp. total) dislocation $O(\log n)$ (resp. $O(n)$), w.h.p.*

5 Derandomization

In Section 4 we showed how it is possible to design a *randomized* algorithm that approximately sorts a sequence S of n elements achieving simultaneously asymptotically optimal maximum dislocation, total dislocation, and running time, w.h.p.⁶ In this section we sketch how `RiffleSort` can be adapted to obtain a *deterministic* algorithm with the same asymptotic guarantees on running time, dislocation, and success probability (over the comparison errors), as long as the order of the elements in S does not depend of the comparison errors.⁷

In order to run `RiffleSort`, we need to partition the input sequence S into a collection of random sets T_0, T_1, \dots, T_k where $k = \frac{\log n}{2}$ and each T_i contains $m = \sqrt{n} \cdot 2^{i-1}$ elements that are chosen uniformly at random from the $n - \sqrt{n} \sum_{j=i+1}^k 2^{j-1} = 2m$ elements in $S \setminus \bigcup_{j=i+1}^k T_j$.

⁶ The *randomized* result also holds when each comparison c has an adversarially chosen and unknown probability of error $p_c \in [0, p]$. The deterministic result holds if $p_c \in [p_0, p]$ for some constant $p_0 > 0$.

⁷ An adversary could make the algorithm fail by first observing all comparison results among the input elements, and then choosing a suitable input permutation S . In other words, our result holds if the comparison errors do not depend on the element values nor on the positions in S of the involved elements.

Notice also that this is the only step in the algorithm that is randomized. To obtain a version of `RiffleSort` that does not require any external source of randomness, i.e., that depends only on the input sequence and on the comparison results, we will generate such a partition by exploiting the intrinsic random nature of the comparison results. As shown in the full version of the paper, with probability at least $1 - \frac{1}{n^3}$, the partition T_0, \dots, T_k can be found in $O(n)$ time using only $6n$ random bits. Moreover, with a technique similar to that of [11], it is possible to simulate “almost-fair” coin flips by xor-ing together a sufficiently large number of comparison results. Indeed, we can associate the two possible results of a comparison with the values 0 and 1, so that each comparison behaves as a Bernoulli random variable whose (unknown) parameter is either p or $1 - p$. We can then use the following fact: let c_1, \dots, c_k be $k = \Theta(\log n)$ independent Bernoulli random variables such that $P(c_i = 1) \in \{p, 1 - p\} \forall i = 1, \dots, k$, then $|\Pr(c_1 \oplus c_2 \oplus \dots \oplus c_k = 0) - \frac{1}{2}| \leq \frac{1}{n^4}$. Therefore, if we consider the set A containing the first $7k$ elements from S and we compare each element in A to all the elements in $S \setminus A$, we obtain a collection of $7k(n - 7k) \geq 6kn$ comparison results (for sufficiently large values of n) from which we can generate $6n$ almost-fair coin flips. A coupling argument shows that, with probability at least $1 - \frac{6kn}{n^4} - \frac{1}{n^3} > 1 - \frac{1}{n^2}$, all these almost-fair coin flips behave exactly as unbiased random bits, and they suffice to select a partition T_0, \dots, T_k of $S \setminus A$. It is now possible to use `RiffleSort` on $S \setminus A$ to obtain a sequence S' having maximum dislocation $d = O(\log n)$ and total dislocation $O(n)$. This requires time $O(n \log n)$ and succeeds with probability at least $1 - |S \setminus A|^{-\frac{3}{2}} > 1 - 3n^{-\frac{3}{2}}$ since $|S \setminus A| \geq \frac{n}{2}$.

What is left to do is to reinsert all the elements of A into S' without causing any asymptotic increase in the total and in the maximum dislocation. While one might be tempted to use the result of Section 1, this is not actually possible since the errors between the elements in A and the elements in S' now depend on the permutation S' . In the full version of this work we show a simple, but slower, $O(n)$ -time strategy to compute $\text{rank}(x, S')$ with an additive error of $O(\log n)$, even when S' is adversarially chosen as a function of the errors. Since A contains $O(\log n)$ elements, simultaneously reinserting them in S' affects the maximum dislocation by at most an $O(\log n)$ additive term, while their combined contribution to the total dislocation is at most $O(\log^2 n)$. We summarize the discussion of this section in the following theorem:

► **Theorem 10.** *Consider a set of n elements that are subject to random persistent comparison errors. For any input permutation of these elements that is chosen independently of the errors, there exists a deterministic algorithm that returns, in $O(n \log n)$ worst-case time, a sequence having maximum (resp. total) dislocation $O(\log n)$ (resp. $O(n)$), w.h.p.*

References

- 1 Miklós Ajtai, Vitaly Feldman, Avinatan Hassidim, and Jelani Nelson. Sorting and selection with imprecise comparisons. *ACM Transactions on Algorithms*, 12(2):19, 2016.
- 2 Laurent Alonso, Philippe Chassaing, Florent Gillet, Svante Janson, Edward M Reingold, and René Schott. Quicksort with unreliable comparisons: a probabilistic analysis. *Combinatorics, Probability and Computing*, 13(4-5):419–449, 2004.
- 3 Michael Ben-Or and Avinatan Hassidim. The Bayesian Learner is Optimal for Noisy Binary Search (and Pretty Good for Quantum as Well). In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 221–230, 2008. doi:10.1109/FOCS.2008.58.
- 4 Mark Braverman, Jieming Mao, and S Matthew Weinberg. Parallel algorithms for select and partition with noisy comparisons. In *Proc. of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 851–862. ACM, 2016.

- 5 Mark Braverman and Elchanan Mossel. Noisy Sorting Without Resampling. In *Proceedings of the 19th Annual Symposium on Discrete Algorithms*, pages 268–276, 2008. [arXiv:0707.1051](#).
- 6 Ferdinando Cicalese. *Fault-Tolerant Search Algorithms - Reliable Computation with Unreliable Information*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2013.
- 7 Yuval Dagan, Yuval Filmus, Daniel Kane, and Shay Moran. The entropy of lies: playing twenty questions with a liar. *CoRR*, abs/1811.02177, 2018. [arXiv:1811.02177](#).
- 8 Peter Damaschke. The Solution Space of Sorting with Recurring Comparison Faults. In *Combinatorial Algorithms - 27th International Workshop, IWOCA 2016, Helsinki, Finland, August 17-19, 2016, Proceedings*, pages 397–408, 2016.
- 9 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with Noisy Information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994. [doi:10.1137/S0097539791195877](#).
- 10 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Sorting with Recurrent Comparison Errors. In Yoshio Okamoto and Takeshi Tokuyama, editors, *28th International Symposium on Algorithms and Computation (ISAAC 2017)*, volume 92 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:12, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. [doi:10.4230/LIPIcs.ISAAC.2017.38](#).
- 11 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Optimal Dislocation with Persistent Errors in Subquadratic Time. In *Proc. of the 35th Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 96 of *LIPIcs*, pages 36:1–36:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. [doi:10.4230/LIPIcs.STACS.2018.36](#).
- 12 Petros Hadjicostas and KB Lakshmanan. Recursive merge sort with erroneous comparisons. *Discrete Applied Mathematics*, 159(14):1398–1417, 2011.
- 13 Rolf Klein, Rainer Penninger, Christian Sohler, and David P. Woodruff. Tolerant Algorithms. In *Proc. of the 19th Annual European Symposium on Algorithm (ESA)*, pages 736—747, 2011.
- 14 Andrzej Pelc. Searching games with errors - fifty years of coping with liars. *Theor. Comput. Sci.*, 270(1-2):71–109, 2002.
- 15 Alfréd Rényi. On a problem of information theory. *MTA Mat. Kut. Int. Kozl. B*, 6:505–516, 1961.
- 16 Ronald L. Rivest, Albert R. Meyer, Daniel J. Kleitman, Karl Winklmann, and Joel Spencer. Coping with Errors in Binary Search Procedures. *J. Comput. Syst. Sci.*, 20(3):396–404, 1980. [doi:10.1016/0022-0000\(80\)90014-8](#).
- 17 Aviad Rubinfeld and Shai Vardi. Sorting from Noisier Samples. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 960–972, 2017. [doi:10.1137/1.9781611974782.60](#).
- 18 Stanislaw M. Ulam. *Adventures of a Mathematician*, 1976.


Dynamic Dominators and Low-High Orders in DAGs

Loukas Georgiadis 

Department of Computer Science & Engineering, University of Ioannina, Greece
loukas@cs.uoi.gr

Konstantinos Giannis

Department of Computer Science & Engineering, University of Ioannina, Greece
giannis_konstantinos@outlook.com

Giuseppe F. Italiano 

LUISS University, Rome, Italy
gitaliano@luiss.it

Aikaterini Karanasiou

Università di Roma “Tor Vergata”, Italy
aikaranasiou@gmail.com

Luigi Laura 

LUISS University, Rome, Italy
llaura@luiss.it

Abstract

We consider practical algorithms for maintaining the dominator tree and a low-high order in directed acyclic graphs (DAGs) subject to dynamic operations. Let G be a directed graph with a distinguished start vertex s . The dominator tree D of G is a tree rooted at s , such that a vertex v is an ancestor of a vertex w if and only if all paths from s to w in G include v . The dominator tree is a central tool in program optimization and code generation, and has many applications in other diverse areas including constraint programming, circuit testing, biology, and in algorithms for graph connectivity problems. A low-high order of G is a preorder of D that certifies the correctness of D , and has further applications in connectivity and path-determination problems.

We first provide a practical and carefully engineered version of a recent algorithm [ICALP 2017] for maintaining the dominator tree of a DAG through a sequence of edge deletions. The algorithm runs in $O(mn)$ total time and $O(m)$ space, where n is the number of vertices and m is the number of edges before any deletion. In addition, we present a new algorithm that maintains a low-high order of a DAG under edge deletions within the same bounds. Both results extend to the case of *reducible* graphs (a class that includes DAGs). Furthermore, we present a fully dynamic algorithm for maintaining the dominator tree of a DAG under an intermixed sequence of edge insertions and deletions. Although it does not maintain the $O(mn)$ worst-case bound of the decremental algorithm, our experiments highlight that the fully dynamic algorithm performs very well in practice. Finally, we study the practical efficiency of all our algorithms by conducting an extensive experimental study on real-world and synthetic graphs.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Connectivity, dominators, low-high orders

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.50

Funding *Giuseppe F. Italiano*: Work partially supported by MIUR, the Italian Ministry for Education, University and Research, under PRIN Project AHeAD (Efficient Algorithms for HARnessing Networked Data).

Acknowledgements We thank the anonymous reviewers for several comments that helped us improve the presentation of our paper.



© Loukas Georgiadis, Konstantinos Giannis, Giuseppe F. Italiano, Aikaterini Karanasiou, and Luigi Laura;

licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 50; pp. 50:1–50:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

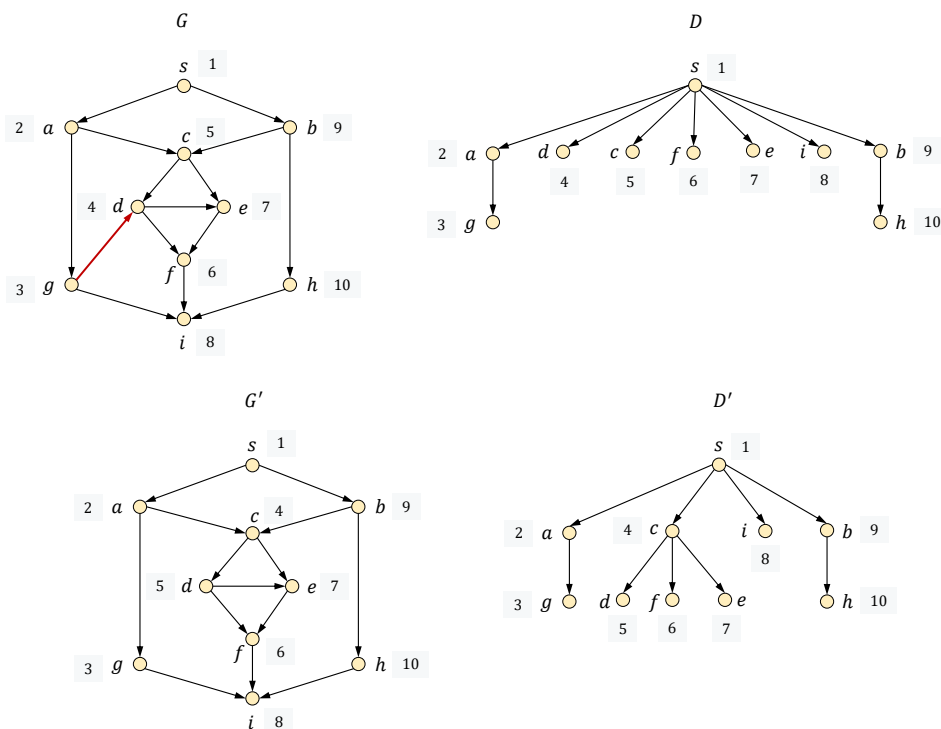
Dynamic graph algorithms have been extensively studied for several decades, and many important results have been achieved for fundamental problems, including connectivity, minimum spanning tree, transitive closure, shortest paths (see, e.g., the survey in [13]). Typically, the goal of a dynamic graph algorithm is to update the solution of a problem, following the insertion or deletion of an edge, as quickly as possible (usually much faster than recomputing from scratch). A dynamic graph problem is said to be *fully dynamic* if it is required to process both insertions and deletions of edges, *incremental* if it requires to process edge insertions only and *decremental* if it requires to process edge deletions only. Here we consider two decremental problems in directed graphs, namely maintaining the dominator tree and a low-high order of a flow graph.

A *flow graph* $G = (V, E, s)$ is a directed graph (digraph) with a distinguished start vertex $s \in V$. A vertex v is *reachable* in G if there is a path from s to v ; v is *unreachable* if no such path exists. The *dominator relation* in G is defined for the set of reachable vertices as follows. A vertex v is a *dominator* of a vertex w (v *dominates* w) if every path from s to w contains v ; v is a *proper dominator* of w if v dominates w and $v \neq w$. The dominator relation in G can be represented by a tree rooted at s , the *dominator tree* D , such that v dominates w if and only if v is an ancestor of w in D . See Figure 1. If $w \neq s$ is reachable, we denote by $d(w)$ the parent of w in D . The dominator tree of a flow graph can be computed in linear time [2, 8, 14, 15]. The dominator tree is a central tool in program optimization and code generation [11], and it has many applications in other diverse areas including constraint programming [40], circuit testing [4], biology [1, 29], memory profiling [38], the analysis of diffusion networks [28], and in connectivity problems [17, 18, 21, 22, 24, 31, 32, 33, 34].

A *low-high order of G* [25] is a preorder of the dominator tree D such that for all reachable vertices $v \neq s$, $(d(v), v) \in E$ or there are two edges $(u, v), (w, v) \in E$, where u and w are reachable, w is not a descendant of v in D , and $u < v < w$ in low-high order. See Figure 1. Every flow graph G has a low-high order, computable in linear-time [25]. Low-high orders provide a correctness certificate for dominator trees that is straightforward to verify [46]. By augmenting an algorithm that computes the dominator tree D of a flow graph G so that it also computes a low-high order of G , one obtains a *certifying algorithm* to compute D . Low-high orders also have applications in path-determination problems [45] and in fault-tolerant network design [5, 6, 26].

In this paper we consider how to maintain the dominator tree and a low-high order of acyclic flow graphs subject to dynamic operations. We believe that acyclic graphs are not a significant restriction, since several real-world networks, such as certain types of biological networks, are acyclic [29]. Furthermore, our results extend to reducible flow graphs (defined below), a class that includes acyclic flow graphs. Reducible flow graphs are important in program optimization since one notion of a “structured” program is that its flow graph is reducible. The dynamic dominator problem arises in various applications, such as data flow analysis and compilation [10, 16]. Moreover, dynamic dominators can be used for dynamically testing various connectivity properties in digraphs, such as 2-vertex connectivity, strong bridges and strong articulation points [32].

The problem of updating the dominator relation has been studied for several decades (see, e.g., [3, 9, 10, 20, 23, 41, 42]). While for the incremental dominators problem there are simple algorithms that achieve total $O(mn)$ running time for processing a sequence of edge insertions in a flow graph with n vertices, where m is the number of edges after all insertions [3, 10, 23], the decremental version seems much harder. Cicerone et al. [10] achieved a total $O(mn)$ update bound for the decremental problem in reducible flow graphs, where



■ **Figure 1** (Top) A flow graph G and its dominator tree D . The numbers correspond to a preorder numbering of D that is a low-high order of G . (Bottom) The flow graph G' and its dominator tree D' after the deletion edge (g, d) .

m is the initial number of edges, but using $O(n^2)$ space. For general digraphs, Georgiadis et al. [20] presented an algorithm that can process a sequence of edge deletions in a flow graph in $O(mn \log n)$ total time and $O(n^2 \log n)$ space, and can answer dominance queries in constant time. For reducible flow graphs, Georgiadis et al. [20] presented an algorithm that achieves $O(mn)$ total running time using only $O(m + n)$ space. A conditional lower bound in [20] suggests that it might be hard to substantially improve the $O(mn)$ update bounds in the partial dynamic (incremental or decremental) problem of maintaining the dominator tree, even for acyclic flow graphs. As the algorithms in [20] are quite sophisticated, their implementation was a challenging task. Nevertheless, we show here that this is really worth the effort, since their efficient implementation performs very well in practice. To produce an implementation of practical value, we performed a careful engineering and choice of data structures, including a data structure for an extension of the dynamic list order maintenance problem [7, 12] and a data structure for maintaining and updating derived edges [25]. To assess the merits of our implementation in practical scenarios, we conducted a thorough experimental study.

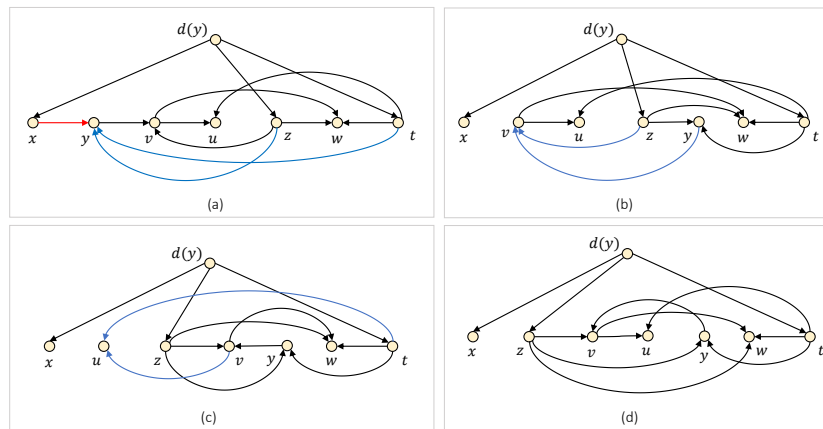
As a second contribution, we show that we can maintain decrementally a low-high order of a reducible flow graph in $O(mn)$ total time. This implies the first decremental *certifying algorithm* [39] for computing dominators in $O(mn)$ total time in reducible flow graphs. It also immediately provides $O(mn)$ -time algorithms for the following problems:

- A data structure that maintains an acyclic flow graph G decrementally, and answers the following queries in constant time: (i) For any two query vertices v and w , find a path π_{sv} from s to v and a path π_{sw} from s to w that are maximally vertex-disjoint, i.e., such that π_{sv} and π_{sw} share only the common dominators of v and w . We can output these

- paths in $O(|\pi_{sv}| + |\pi_{sw}|)$ time. (ii) For any two query vertices v and w , find a path π_{sv} from s to v that avoids w , if such a path exists. We can output this path in $O(|\pi_{sv}|)$ time. Such a data structure (in the static case) was used by Tholey [45] in a linear-time algorithm for the 2-disjoint paths problem in a directed acyclic graph (DAG).
- A decremental version of the fault-tolerant reachability problem [5, 6] in DAGs. We maintain an acyclic flow graph $G = (V, E, s)$ through a sequence of edge deletions, so that we can answer the following query in $O(n)$ time. Given a spanning forest $F = (V, E_F)$ of G rooted at s , find a set of edges $E' \subseteq E \setminus E_F$ of minimum cardinality, such that the subgraph $G' = (V, E_F \cup E', s)$ of G has the same dominators as G .

An incremental low-high order algorithm with $O(mn)$ total update time was presented in [19]. As in the dynamic dominators problem, the decremental version seems more difficult than the incremental. To highlight this aspect, note that a single edge deletion can cause $O(n)$ changes in a given low-high order even if the dominator tree remains unaltered. See Figure 2. On the other hand, in the incremental setting, it suffices to update the low-high order only for the vertices that change parent in the dominator tree.

Our third contribution is an efficient fully dynamic algorithm for maintaining the dominator tree of a DAG under an intermixed sequence of edge insertions and deletions. We obtain this algorithm by incorporating the insertion method of [23] in our decremental algorithm. The fully dynamic algorithm does not preserve the $O(mn)$ worst case bound of the decremental algorithm because the vertex depths in the dominator tree no longer change monotonically. Despite this, however, our experimental results show that it performs very well in practice.



■ **Figure 2** An example of propagation of changes in the low-high order after the deletion of an edge. Vertices are arranged from left to right in low-high order. (a) After the deletion of (x, y) , y violates the given low-high order. (b)-(c) Moving y between z and t causes a new violation at vertex v , which in turn causes another violation at vertex u after v is placed between z and y . (d) The low-high order is finally restored when we place u between v and t .

2 Preliminaries

Let $G = (V, E, s)$ be a flow graph with start vertex s , and let D be the dominator tree of G . For any vertex $v \in V$, we let $In(v)$ denote the set of vertices that have an edge in G entering v , i.e., $In(v) = \{u \in V : (u, v) \in E\}$. An edge (x, y) of flow graph G is a *bridge* if its deletion makes y unreachable from s . Given a rooted tree T , we denote by $T(v)$ the subtree of T rooted at v (we also view $T(v)$ as the set of descendants of v). Let T be a tree rooted at s

with vertex set $V_T \subseteq V$, and let $t(v)$ denote the parent of a vertex $v \in V_T$ in T . If v is an ancestor of w , $T[v, w]$ is the path in T from v to w . In particular, $D[s, v]$ consists of the vertices that dominate v . If v is a proper ancestor of w , $T(v, w]$ is the path to w from the child of v that is an ancestor of w . Analogously, $T[v, w)$ denotes the path from v to $t(w)$. Suppose now that the vertex set V_T of T consists of the vertices reachable from s . Tree T has the *parent property* if for all $(v, w) \in E$ with v and w reachable, v is a descendant of $t(w)$ in T . If T has the parent property and has a low-high order, then $T = D$ [25]. For any vertex $v \in V$, we denote by $C(v)$ the set of children of v in D . A *preorder* of T is a total order of the vertices of T such that, for every vertex v , the descendants of v are ordered consecutively, with v first. A preorder of D is a low-high order of G , if $(d(v), v) \in E$ or there are two edges $(u, v), (w, v) \in E$ such that $u < v < w$, and w is not a descendant of v in D .

A *reducible* flow graph [30, 44] is one in which every strongly connected subgraph S has a single *entry* vertex $v \in S$ such that every path from s to a vertex in S contains v . A flow graph is reducible if and only if it becomes acyclic when every edge (v, w) such that w dominates v is deleted [44]. We refer to such an edge as a *back edge*. Deletion of such edges reduces the problem of computing dominators on a reducible flow graph to the same problem on an acyclic graph.

3 Decremental dominators

The algorithm of Georgiadis et al. [20] is based on the concept of *derived edges*. Recall that from the parent property of D , for any edge (v, w) of G , $d(w)$ is an ancestor of v in D . Let (v, w) be an edge of G , with w not an ancestor of v in D . (Such edges do not exist if G is acyclic.) Then, the *derived edge* of (v, w) is the edge (\bar{v}, w) , where $\bar{v} = v$ if $v = d(w)$, \bar{v} is the sibling of w in D that is an ancestor of v if $v \neq d(w)$. If w is an ancestor of v in D , then the derived edge of (v, w) is null. Note that a derived edge (\bar{v}, w) may not be an original edge of G . Given the dominator tree D of a flow graph $G = (V, E, s)$ and a list of edges $S \subseteq E$, we can compute the derived edges of S in $O(|V| + |S|)$ time [25].

Now consider the effect of an edge deletion on the dominator tree D . Let (x, y) be the deleted edge. We call the deletion of (x, y) *regular* if (x, y) is not a bridge of G , i.e., y remains reachable from s after the deletion. We let G' and D' denote the flow graph and its dominator tree after the update ($G' = G \setminus (x, y)$). Similarly, for any function f on V , we let f' be the function after the update. In particular, $d'(v)$ denotes the parent of v in D' . By definition, $D' \neq D$ only if x is reachable before the update. We say that a vertex v is *affected* by the update if $d'(v) \neq d(v)$, and *unaffected* otherwise. If v is affected then $d'(v)$ does not dominate v in G . Since the effect of an edge deletion is the reverse of an edge insertion, [23, Lemma 1], and [25, Lemma 4.1] imply the following:

► **Lemma 1.** *Suppose x is reachable and the deletion of edge (x, y) is regular, i.e., y does not become unreachable after the deletion. Then the following statements hold:*

- (a) *All affected vertices become descendants in D' of a child c of $d(y)$.*
- (b) *A vertex v is affected if and only if $(d(v), v)$ is not an edge of G' and all edges $(u, v) \in E \setminus (x, y)$ correspond to the same derived edge $(\bar{u}, v) = (c, v)$ of G .*
- (c) *After the deletion, each affected vertex v becomes a child of a vertex on the critical path $D'[c, d'(y)]$.*
- (d) *No vertex on $D'[c, d'(y)]$ is affected. Hence, $D'[c, d'(y)] = D[c, d'(y)]$.*

We note that statements (a) and (c) hold for arbitrary flow graphs, while (b) and (d) are true only for acyclic (and reducible) flow graphs. The algorithm of [20] applies Lemma 1 in order to locate the affected vertices in some topological order of G as follows. For each vertex

v we maintain a count $InSiblings(v)$ which corresponds to the number of distinct siblings w of v such that (w, v) is a derived edge. We also maintain the lists $DerivedOut(v)$ of the derived edges (v, u) leaving each vertex v . As we locate each affected vertex, we find its new parent in the dominator tree and update the counts $InSiblings$ for the siblings of v . The first step is to update $InSiblings(y)$. If $InSiblings(y) = 1$, then we compute the nearest common ancestor $z = d'(y)$ in D' of all vertices in $In(y)$. In this case, by Lemma 1(c), z is a descendant of a sibling c of y in D . Next, we update the $InSiblings(v)$ counts for all $v \in DerivedOut(y)$. Specifically, we decrement $InSiblings(v)$ if $v \in DerivedOut(c)$; if $InSiblings(v) = 1$ then we identify v as affected and inserted into a FIFO queue Q . Then we repeat the same process for each vertex extracted from Q . We can locate the new parent $d'(v)$ of each affected vertex v in D' as for y , i.e., by computing the nearest common ancestor in D' of all vertices in $In(v)$. This way, however, does not guarantee the desired $O(mn)$ total update time. Therefore, we locate $d'(v)$ by traversing the critical path $D[c, d'(y)]$ in top-down order, until we find a vertex u such that $In(v)$ contains a vertex that is not a descendant of u in D' . Then we have $d'(v) = d(u)$. Finally, we can compute the updated $InSiblings$ counts and $DerivedOut$ lists in a postprocessing step. The analysis in [20] is based on the fact that the affected vertices that remain reachable increase their depth in D .

3.1 Efficient implementation

Providing a practical version and an efficient implementation of the above algorithm turns out to be a very challenging task. In particular, we need to incorporate efficient solutions to the following subproblems: (i) answering ancestor-descendant queries in the dominator tree D that changes dynamically, (ii) maintaining dynamically the derived edges of G , and (iii) handling the deletion of bridges. We note that (i) and (ii) are not needed when we update D incrementally.

Ancestor-descendant queries. Throughout the execution of the deletion sequence, we need to test in $O(1)$ time the ancestor-descendant relation between pairs of vertices in D , in order to locate the new parent of each affected vertex $v \neq y$. To that end, it suffices to recompute a preorder and a postorder numbering of the vertices in D after each update, since this takes $O(n)$ time by simply performing a dfs traversal of D . Then, v is a descendant of u in D if and only if $u \leq v$ in preorder and $v \leq u$ in postorder [43]. Another option is to represent each order (preorder and postorder) with a data structure for the dynamic list order problem [7, 12]. Both methods guarantee the desired $O(mn)$ total update bound, but the use of a dynamic list order data structure gives a much faster implementation in practice.

Here, we also take advantage of the fact that for each affected vertex v we can move the entire subtree of $D(v)$ in the new location in the dynamic lists, rather than inserting the vertices in $D(v)$ one by one. Specifically, we remove the subtree $D(v)$ from its current locations in the two dynamic lists and insert them immediately after $d'(v)$ in the preorder list and immediately before the first descendant of $d'(v)$ in the postorder list. Thus, we have immediate access to the appropriate location in the preorder list, but we still need to find the corresponding location in the postorder list. In order to do this search fast, we maintain, for each $v \in D$, the list $C(v)$ of the children of v in D ordered in preorder. Then, we can find the first descendant of a vertex u in the postorder list by repeatedly following the links to the first child in preorder, until we reach a leaf.

We implemented the dynamic preorder and postorder lists by adapting the dynamic list order data structure of Bender et al. [7] that uses a two-level structure (implementing a numbering scheme) and supports insertions, deletions and order queries in constant amortized time. We extend this structure so that it can also support the following operation:

move(u, v, w): Move the items between u and v (inclusive) from their current location in the dynamic list and insert them right after w .

We implement the above operation as follows. First, we find the representative nodes (in the top-level structure) for u and v . We check if the left-representative (right-representative, respectively) has items in the second-level list that do not belong to the moved set of items; if there are such items then we split the second-level lists and create new representative items. After this step, both representative items and every other representative item between them, have only second-level items that belong to the set we want to move. Hence, we can quickly move the entire set by linking the left-representative and right-representative to their new position in the dynamic list, right after w . Finally, we check if we can merge the representatives that we move or split with their neighbours.

In our experiments, we observed that the above method was several orders of magnitude faster compared to recomputing a preorder and a postorder numbering in D .

Derived edges. Recall that after finding the affected vertices of an edge deletion, we need to compute the updated *InSiblings* counts and *DerivedOut* lists. The only types of edges that may change their corresponding derived edge are (i) edges entering affected vertices, and (ii) edges that enter a former sibling of y from a descendant of an affected vertex. Let S be the set of these edges. As mentioned above, we can compute the derived edges of S in $O(n + |S|)$ time [25], which suffices for our $O(mn)$ bound since every edge in S is adjacent to at least one vertex that changes depth in D . The method given in [25] for computing derived edges is based on bucket sorting using a preorder numbering of D . This is not suitable for our framework, since we do not maintain a preorder numbering of the vertices, but use a dynamic list order data structure instead. Here we propose a more practical method. First we note that for each edge (u, v) of type (ii), i.e., u is a descendant of an affected vertex and $d(v) = d'(v) = d(y)$, we have $\bar{u} = c$. Now let (u, v) be of type (i), i.e., v is affected so $d'(v) \in D'[c, y]$ and u is a descendant of $d'(v)$. If $u = d'(v)$ then $\bar{u} = u$, so suppose u is a proper descendant of $d'(v)$. Let w_v be the next vertex on $D'[c, y]$ following $d'(v)$ ($w_v = d'(d'(v))$), and let z_u be the nearest ancestor of u such that $d'(z_u) \in D'[c, y]$. Then, $\bar{u} = w_v$ if $d'(z_u) \neq d'(v)$, and $\bar{u} = z_u$ if $d'(z_u) = d'(v)$. Note that we have already computed w_v , for each affected vertex v , when we locate its new parent in D' . Hence, it suffices to compute z_u for all edges (u, v) where u is a proper descendant of $d'(v)$. We do that by visiting the ancestors of u until we reach z_u . First we mark all vertices on $D'[c, y]$, so we stop our search when reaching a vertex that has a marked parent. To avoid multiple visits to the same vertices, we maintain at each vertex w a label $l(w)$, initially null. After we locate z_u , we set $l(w) = z_u$ for each visited vertex w . Thus, the next search stops at a vertex w such that $d'(w)$ is marked or $l(w)$ is not null. Therefore, we can compute all the new derived edges in $O(n + |S|)$ time as desired.

Unreachable vertices. After the deletion of an edge (x, y) , some vertices may become unreachable. This happens when (x, y) is a bridge of the current flow graph. Since we deal with acyclic graphs, this means that (x, y) is the only edge entering y from a reachable vertex. Hence, we can detect easily if (x, y) is a bridge, since we have $InSiblings(y) = 0$ and $d(y) = x$. In order to achieve $O(mn)$ total running time, we can simply recompute the dominator tree from scratch after each such deletion, since the total number of bridges that can appear is at most $n - 1$. In practice, however, this causes a significant slowdown of our algorithm. A better idea is to handle the deletion of a bridge (x, y) as follows:

1. Compute the set of edges Y from vertices in $D(y)$ to vertices in $D \setminus D(y)$. Note that no edge $e \in Y$ is a bridge in $G \setminus (Y \setminus e)$, since for any vertex $v \in D \setminus D(y)$, all edges in $(w, v) \in Y$ correspond to the same derived edge (\bar{w}, v) .
2. Process each edge $e \in Y$ as a regular deletion.
3. Delete $D(y)$ from the dominator tree D' of G' , and update accordingly the data structures. Note that Steps 1 and 3 take $O(m)$ time. Also, since in Step 2 we have regular deletions, the total running time remains $O(mn)$.

4 Decremental low-high order

Now we consider how to update a low-high order of an acyclic flow graph $G = (V, E, s)$ after the deletion of an edge (x, y) . First, we show how to achieve an $O(mn)$ total update bound using a sparsification technique, similar to the one used for the incremental problem in [19]. The idea is to maintain a subgraph $H = (V, E_H)$ of G with $O(n)$ edges that has the same dominator tree as G . By Lemma 1(c), each vertex v with $(d(v), v) \notin E$ has two entering edges (u, v) and (w, v) such that $\bar{u} \neq \bar{w}$; then, it suffices to add two such edges in H .

► **Corollary 2.** *Let $H = (V, E_H)$ be subgraph of an acyclic flow graph G such that E_H contains:*

- (a) *All edges $(u, v) \in E$ such that $u = d(v)$.*
- (b) *Two edges (u, v) and (w, v) such that $\bar{u} \neq \bar{w}$ for each vertex v with $(d(v), v) \notin E$.*

Then H has the same dominator tree as G . Moreover, a low-high order of H is also a valid low-high order of G .

Note that the two edges in Corollary 2(b) exist by Lemma 1(c). Clearly $H = (V, E_H)$ has $O(n)$ edges as required. Now, we can compute a low-high order of H in $O(|E_H|) = O(n)$ time using the static algorithm of [25]. The algorithm arranges the children $C(x)$ of each non-leaf vertex x of D in a local low-high order δ_x . First, we place all vertices $v \in C(x)$ with $(x, v) \in E$ in arbitrary order in δ_x . Then, we process the remaining children of x in topological order. For each such vertex v , H contains edges (u, v) and (w, v) such that $\bar{u} \neq \bar{w}$, so \bar{u} and \bar{w} precede v in the topological order and are already located in δ_x . Hence, it suffices to insert v in any location in δ_x between \bar{u} and \bar{w} . When we have computed all the local low-high orders, we can get the complete low-high order of G by arranging each subtree $D(v)$ of D immediately after v . After the deletion of (x, y) we need to update H in order to ensure that it still satisfies Corollary 2. We can do this during the update of the derived edges, after we have located all their affected vertices and their new parents in D' . Therefore, we get the following result.

► **Theorem 3.** *We can maintain a low-high order of a reducible flow graph G with n vertices through a sequence of edge deletions in $O(mn)$ total time, where m is number of edges in G before all deletions.*

4.1 Bounded search algorithm

Here we present an algorithm that updates a low-high order much faster in practice than the above algorithm. To that end, we also need to maintain the lists $DerivedIn(v)$ of the derived edges (u, v) entering each vertex v . The algorithm is based on two ideas. First, we observe that it is easy to update the low-high order for the affected vertices. The problematic case is to update the low-high order for unaffected vertices. For the latter case, we propose a bounded search process that identifies the vertices that may need to be relocated in low-high order.

Affected vertices. The crucial observation is that the decremental dominators algorithm of Section 3 discovers the affected vertices in topological order. Thus, after we move all the affected vertices in their new locations in D' and update their incoming derived edges, we can position them in low-high order. For each affected vertex v , if $(d(v), v) \notin E$, then $DerivedIn(v)$ contains two vertices u and w such that $u < w$ in low-high order, so we can insert v between these two vertices.

Unaffected vertices. Now we deal with the more challenging case of updating the low-high order of unaffected vertices. As we already observed, a single edge deletion may cause many changes in a given low-high order, even if there are no affected vertices. (See Figure 2.) Our first step is to identify the initial set I of unaffected vertices that violate the low-high order after updating the dominator tree and the low-high order of the affected vertices. Fixing the low-high order of the vertices in I may invalidate the low-high order of other vertices that are reachable from I . So, our next step is to compute a set X ($I \subseteq X$) of vertices that may need to be relocated in low-high order due to the changes in the low-high order of I . The next lemma determines the location of the vertices in I .

► **Lemma 4.** *Let v be an unaffected vertex that violates the given low-high order after updating the dominator tree in response to an edge deletion (i.e., $v \in I$). Then $d'(v) = d(y)$.*

Proof. A vertex v may violate the low-high order only if it has an entering edge (u, v) such that u is a descendant of an affected vertex and the derived edge of (u, v) changes. From the parent property of the dominator tree we have that for all $(v, w) \in E$ with v and w reachable, v is a descendant of $d(w)$ in D . Since, by Lemma 1(c), all affected vertices become descendants of a child c of $d(y)$, the derived edge of (u, v) changes only if v is a child of $d(y)$. Since v is unaffected, $d'(v) = d(v) = d(y)$. ◀

The above lemma also helps us limit our search for candidate vertices that may need to be relocated in the low-high order in response to the update of the position of the vertices in I . Since I consists only of children of $d(y)$, we only need to search among the unaffected children of $d(y)$ that are reachable from I . As we relocate vertices in low-high order, this process may cascade. See Figure 2.

In order to bound the total running time of our algorithm by $O(mn)$, we maintain a sparse spanning subgraph $H = (V, E_H)$ of G with $O(n)$ edges that satisfies Corollary 2, together with the derived edges \overline{E}_H of E_H . We also maintain the invariant that for each vertex v such that $(d(v), v) \notin E$, the two derived edges $(u, v), (w, v) \in \overline{E}_H$ are such that $u < v < w$ in low-high order.

Our algorithm, $FixLH(y)$, computes a set of vertices $X \subseteq C'(d(y))$ that we will need to process in order to ensure that they satisfy a low-high order of G' . Initially, we set $X = I$ and execute a search from each vertex in I in order to discover vertices that may violate the given low-high order due the replacement of the vertices in I . During this search, we would like to avoid any unnecessary propagation of changes in the low-high order. To achieve this, when we process a vertex $u \in X$, we examine its outgoing derived edges in \overline{E}_H . For each such edge (u, v) we test if v satisfies the current low-high order without considering the derived edges from X . If this is not the case, then we insert v into X . This bounded search is outlined by Procedure `scan`. Note that we can only afford to check a constant number k of entries in $DerivedIn(v)$ in order to have $O(n)$ running time per deletion. (In our experiments we set $k \leq 3$.) Thus, we get the following result.

► **Lemma 5.** *Algorithm $FixLH$ correctly updates the low-high order of the children of $d(y)$ in D' in $O(n)$ time.*

Algorithm 1 FixLH(y).

```

1  $I =$  children of  $d(y)$  that violate the low-high order of  $G$  after the deletion  $/*I \subseteq \{y\}$ 
   if  $y$  is not affected; otherwise,  $I$  contains unaffected children of  $d(y)$  that
   have an entering edge from a descendant of an affected vertex */
2 initialize  $X = I$   $/*X$  will contain the unaffected children of  $d(y)$  that need to be
   relocated in low-high order */
3 foreach vertex  $u \in I$  do
4   | if  $u$  not scanned then scan( $u$ )
5 end
6 Process vertices in  $X$  in topological order to place them in low-high order using the
   edges in  $\overline{E}_H$ 

```

Procedure scan(u).

```

1 foreach derived edge  $(u, v) \in \overline{E}_H$  do
2   | if  $v \notin X$  and  $(d(v), v) \notin E$  then
3     | if  $u < v$  in low-high order then
4       | examine the first  $k = O(1)$  edges in  $DerivedIn(v)$  to find a replacement
5         | derived edge  $e = (w, v)$  with  $w \notin X$  and  $w < v$  in low-high order
6       | end
7     | else
8       | examine the first  $k = O(1)$  edges in  $DerivedIn(v)$  to find a replacement
9         | derived edge  $e = (z, v)$  with  $z \notin X$  and  $v < z$  in low-high order
10      | end
11     | if a replacement derived edge  $e$  was found then
12       | replace  $(u, v)$  with  $e$  in  $\overline{E}_H$ 
13     | end
14     | else
15       | insert  $v$  into  $X$ 
16       | scan( $v$ )
17     | end
18   | end
19 end

```

Proof. To prove the correctness of algorithm FixLH, first note that it correctly updates the low-high order of all vertices in X . Now we need to argue that the remaining vertices satisfy the updated low-high order. Observe that any vertex v that is visited during the search for X , is not inserted into X only if $(d(v), v) \in \overline{E}_H$ or if both derived edges in \overline{E}_H entering v are not in X . Clearly, the same holds for all vertices that are not visited during this process. Hence, any vertex $v \notin X$ does not violate the computed low-high order before and after relocating the vertices in X .

Now we argue that the algorithm runs in $O(n)$ time. Each vertex v may change its two entering edges in \overline{E}_H at most $O(1)$ times, since we look for replacement edges only in the first $O(1)$ edges in $DerivedIn(v)$. Thus, $DerivedIn(v)$ will be examined in lines 4 and 7 of Procedure scan a constant number of times in total for each v , so we spend constant time for

each vertex. Finally, we need to process the vertices of X in topological order. Note that the vertices may be inserted in X in arbitrary order. We can sort them topologically by computing a topological order of the subgraph of $\overline{H} = (V, \overline{E}_H)$ that is induced by the vertices of X . Since \overline{E}_H has $O(n)$ edges, this step also takes $O(n)$ time. ◀

► **Remark 6.** We also test the following, slightly more complicated, variant of Algorithm FixLH. In line 6, where we place each vertex of $u \in X$ in low-high order, we do not use only the derived edges entering u that are contained in \overline{E}_H , but also consider a constant number of derived edges entering u contained in $\overline{E} \setminus \overline{E}_H$. Let $\overline{E}(u)$ be the set of derived edges entering u that we consider in order to decide the location of u in the low-high order. Also, let $\overline{V}(u) = \{w \in V : (w, u) \in \overline{E}(u)\}$, i.e., the vertices from which the derived edges in $\overline{E}(u)$ originate. Then, we place u right after the median vertex in $\overline{V}(u)$, sorted with respect to the low-high order. By doing this placement for u , we hope that Procedure scan will have better chances for locating replacement derived edges.

4.2 Implementation issues

We extend our decremental dominators algorithm of Section 3 so that it also maintains a low-high order as described above. The following implementation issues affect the efficiency of our algorithm in practice.

Representation of a low-high order. Since a low-high order is a preorder of D , we can use the same dynamic list order data structure as in Section 3.1. This choice, however, has the serious drawback that we may need to update the data structures for both the preorder and the postorder of D much more often than in Section 3.1. For this reason, we use a separate dynamic list order data structure for the low-high order, which is updated independently of the preorder and the postorder of D .

Unreachable vertices. As in the decremental dominators algorithm of Section 3.1, we have to take special care of how the deletion of a bridge (x, y) is handled. To that end, we first tested the two methods mentioned in Section 3.1: (a) Run a static algorithm to recompute the dominator tree D and a low-high order from scratch, and (b) Process each edge $e = (u, v)$ with u a descendant of y in D and v not a descendant of y in D as a regular deletion (e cannot be a bridge) and update the low-high order after each such deletion. Then delete (x, y) , making all descendants of y in D unreachable from s .

Unlike the decremental dominators algorithm, choice (b) here is not always superior to (a) because during the sequence of regular deletions a vertex may be scanned several times when the FixLH process is executed. Hence, we also implemented the following improvement, which updates the low-high order of unaffected vertices after the sequence of regular deletions is processed. Specifically, we first update the dominator tree as in (b) but do not compute the complete low-high order after each regular deletion of an edge $e = (u, v)$. As we process each regular deletion (u, v) , we also fix the low-high order of each affected vertex. Let A^* denote the set of all affected vertices found during all regular deletions. For each edge (w, t) such that w is a descendant of an affected vertex in A^* we insert t in a list I^* . We compute a set X^* of vertices which may need their low-high to be updated by executing $scan(v)$, starting from all vertices v in I^* that have not been scanned yet. Finally, we sort X^* topologically and update the low-high order of all vertices in X^* .

All of the above three methods are executed in $O(m)$ time per bridge deletion, so they all guarantee the $O(mn)$ total running time. In our experiments, however, the last method turned out to be an order of magnitude faster than (a) and (b).

5 Empirical Analysis

We wrote our implementations in C++, using g++ v.4.6.4 with full optimization (flag `-O3`) to compile the code. We report the running times on a Dell Precision Tower 7820 CTO Base machine running Ubuntu (16.04 LTS), equipped with an Intel Xeon Gold 5118 2.3 GHz processor with 16 MB L3 cache and 192GB DDR4-2400 RAM at 2,666 MHz. We did not use any parallelization, and each algorithm ran on a single core. We report CPU times measured with the `getrusage` function.

■ **Table 1** Graph instances used in the experiments. The original graphs are taken from [35] and [37], and were converted to DAGs by including vertices and edges reachable from the start vertex and deleting depth-first search back edges. The graph categories are: source code (SC), social network (SN), peer to peer network (P2P), web graph (WG), communication network (CN), and product co-purchasing network (PN). The number of vertices n and edges m refer to the produced instances.

Graph	Graph Details			Reference
	Type	n	m	
linux	SC	1524	3687	KONECT [35]
advogato	SN	2320	17809	KONECT [35]
p2p-Gnutella31	P2P	14149	32363	SNAP [37]
Amazon0302	PN	55414	126663	SNAP [37]
Soc-Epinions1	SN	17117	158754	SNAP [37]
web-BerkStan	WG	29145	169870	SNAP [37]
WikiTalk	CN	49430	664139	SNAP [37]
Amazon0601	PN	276049	1259198	SNAP [37]
web-Google	WG	600493	2013471	SNAP [37]

Table 1 shows some statistics about the graphs used in our experimental evaluation. In all test instances we select the first vertex of the graph as the start vertex. (Choosing a random start vertex produces similar results.) We produce decremental instances as follows. The number of edges that will be deleted is controlled by a parameter $p \in [0, 1]$. Let m be the initial number of edges in the graph. We create a sequence of deletions by choosing $\lfloor p \cdot m \rfloor$ edges in the original graph uniformly at random. For each graph and each choice of p , we create 10 such random instances using different seeds for the initialization of the random functions, and report the average running times. (For a given input graph, two values $p_1 < p_2$ of p , and a fixed seed, the deletion sequence for p_1 is a subsequence of the deletion sequence for p_2 .) The algorithms compute (in static mode) the dominator tree (and a low-high order for the decremental low-high order algorithms) of the original graph and then they run in decremental mode, processing the sequence of deletions. Note that during the execution of the algorithms some vertices may become unreachable, and thus some subsequent deletions may involve a disconnected portion of the graph. These deletions are detected and ignored by all algorithms. For computing dominators in static mode we use the SNCA algorithm from [27], which is a simplified variant of the classic Lengauer-Tarjan algorithm [36] that performs very well in practice. (Algorithm SNCA was generally faster than other well-known algorithms tested in [27].) As an intermediary, this algorithm computes a sparse subgraph H of the input graph G that has the same dominators as G . The indegree of each vertex in H is at most 2, so H has at most $2(n - 1)$ edges (the start vertex has zero indegree). For computing a low-high order, we augment this algorithm with the low-high order algorithm for acyclic graphs from [25]. We speedup the computation of a low-high order by using only the edges in H (instead of all the edges of G).

Dominators. We compare our efficient algorithm, *Decr* of Section 3, with two dynamized versions of *SNCA*. (We did not consider the algorithm of Cicerone et al. [10] since it requires $O(n^2)$ space, and therefore is impractical for large graphs.) The first algorithm, *DSNCA1*, first tests if the deleted edge (x, y) belongs to the sparse subgraph H . If not, then the dominator tree is not affected and the algorithm does nothing. Otherwise, it simply runs *SNCA* from scratch. The second algorithm, *DSNCA2*, also performs the same test, but if $(x, y) \in H$, then it computes the nearest common ancestor z of x and y in D and runs *SNCA* only for the subgraph of G induced by $D(z)$.

The average running times are reported in Table 3. Our experimental results show that the new algorithm *Decr* is much faster than both *DSNCA1* and *DSNCA2*. In particular, *Decr* is consistently at least two orders of magnitude faster than *DSNCA1* and *DSNCA2*. For several graphs considered in our experiments, a large fraction of the vertices tended to get disconnected from the start vertex after the deletion of about 50% of the edges, and therefore many subsequent deletions are ignored. We also observe that recomputing the dominator tree only for the subgraph induced by $D(z)$ (where z is the nearest common ancestor of x and y in D) does not provide a significant improvement in the running time of *DSNCA*, and it may even cause slowdown in some instances, due to the overhead of computing the nearest common ancestor of x and y in D .

Fully-dynamic case. We also report some experimental results for our fully dynamic algorithm, that we refer to as *Dyn*, that maintains the dominator tree of a DAG under a mixed sequence of edge insertions and deletions. We obtain this algorithm by incorporating the insertion method of [23] in our decremental algorithm. Note that, as in our decremental algorithm, we need to maintain the same data structures for the derived edges and for the dynamic preorder and postorder lists. (These data structures are not required in the incremental setting.) The fully dynamic algorithm does not preserve the $O(mn)$ worst case bound of the incremental or the decremental algorithm because the vertex depths in the dominator tree no longer change monotonically. Despite this, however, our experimental results show that it performs very well in practice.

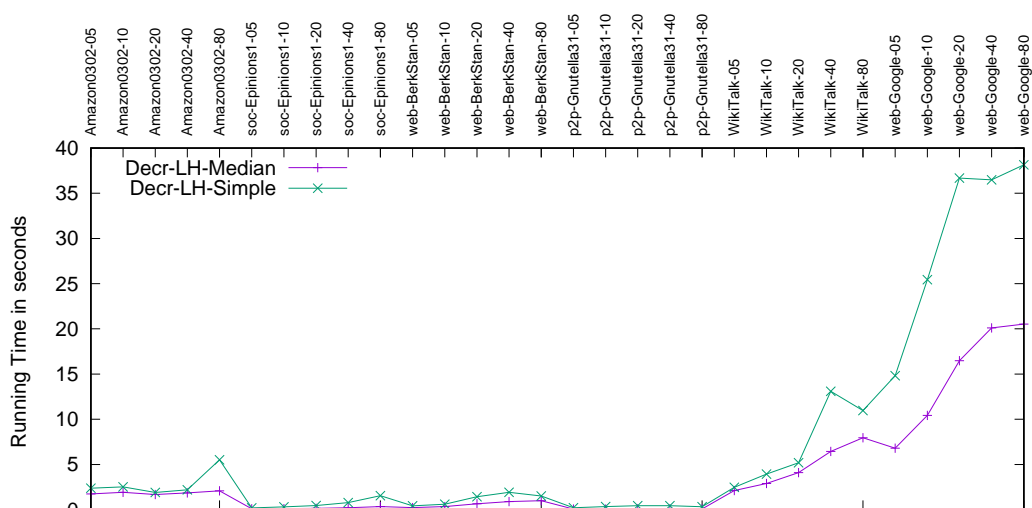
We produce fully dynamic instances as follows. The number of edges that will be inserted and deleted are controlled by parameters $p_i \in [0, 1]$ and $p_d \in [0, 1]$, respectively. Let m be the initial number of edges in the graph. We create a sequence of insertions and deletions by choosing $m_i = \lfloor p_i \cdot m \rfloor$ random edge insertions and $m_d = \lfloor p_d \cdot m \rfloor$ random edge deletions. This means that the flow graph initially has $m' = m - m_i$ edges. For each graph and each choice of p_i, p_d , we create 10 such random instances using different seeds for the initialization of the random functions, and report the average running times. The algorithms compute (in static mode) the dominator tree of the original graph and then they run in fully-dynamic mode, processing the (intermixed) sequence of insertions and deletions. As in the decremental algorithm, we compute the dominator tree in static mode with the *SNCA* algorithm from [27]. In the fully-dynamic mode, the type of each update operation is chosen uniformly at random, so that there are m_i insertions interspersed with m_d deletions. During this simulation that produces the dynamic graph instance we keep track of the edges currently present in the graph. If the next operation is a deletion then the edge to be deleted is chosen uniformly at random from the edges in the current graph.

Here we only give some preliminary experimental results, presented in Table 2, and defer to the full version of the paper for a more comprehensive experimental study. As in the decremental setting, we observe significant speed ups with respect to *DSNCA1* and *DSNCA2*.

■ **Table 2** Average running times in seconds over 10 random intermixed update sequences of edge insertions and deletions. The suffixes in the graph names correspond to the percentage of inserted edges and deleted edges, respectively.

Graph_insertion_deletion	Fully Dynamic Dominators		
	DSNCA1	DSNCA2	Dyn
soc-Epinions1_10_10	11.720	9.072	0.248
soc-Epinions1_30_40	10.620	8.956	0.228
soc-Epinions1_40_20	10.696	7.396	0.268
Amazon0302_30_40	0.900	1.424	0.104
web-BerkStan_30_40	0.156	0.284	0.080
web-BerkStan_40_20	0.992	1.584	0.072

Low-high order. Here we examine the efficiency of our algorithm Decr-LH, using the slightly more complicated variant for placing of a vertex in low-high order, described in Remark 6. In some instances, this variant performed significantly better than the simple version, as shown in Figure 3. We compare the running time of Decr-LH with a dynamized version of SNCA that also computes a low-high order of an acyclic flow graph. This algorithm, that we refer to as DSNCA-LH, works as follows. It maintains a sparse subgraph $H = (V, E_H)$ of G such that for each $v \neq s$, $(d(v), v) \in E_H$, or E_H contains edges (u, v) and (w, v) with $u < v < w$. When we delete an edge (x, y) we test if this edge belongs to H . If not, then the dominator tree and the low-high order are not affected, so we do nothing. Otherwise, we look into the entering edges of v and try to find a replacement edge for (x, y) so that y satisfies the current low-high order. If this fails, then we compute the dominator tree and the low-high from scratch.



■ **Figure 3** Running time of two implementations of Decr-LH. The first, Decr-LH-Simple, uses the simple method for placing of a vertex in low-high order, as described in Algorithm FixLH, while the second, Decr-LH-Median, uses the method described in Remark 6.

■ **Table 3** Average running times in seconds over 10 random deletion sequences. The suffixes in the graph names correspond to the percentage of deleted edges $p = 5\%, 10\%, 20\%, 40\%$, and 80% . Running times exceeding 2.5 hours are not reported.

Graph_deletion	Decremental Dominators			Decremental Low-High	
	DSNCA1	DSNCA2	Decr	DSCNA-LH	Decr-LH
linux_05	0.060	0.056	0.001	0.272	0.001
linux_10	0.108	0.116	0.001	0.604	0.001
linux_20	0.228	0.212	0.004	0.974	0.004
linux_40	0.428	0.396	0.004	1.488	0.004
linux_80	0.620	0.640	0.004	2.212	0.004
advogato_05	0.084	0.084	0.008	0.760	0.008
advogato_10	0.080	0.080	0.004	0.828	0.012
advogato_20	0.084	0.080	0.004	0.780	0.008
advogato_40	0.084	0.080	0.008	0.684	0.008
advogato_80	0.084	0.080	0.008	0.784	0.008
p2p-Gnutella31_05	0.916	0.788	0.020	4.648	0.064
p2p-Gnutella31_10	1.924	1.712	0.032	4.912	0.072
p2p-Gnutella31_20	2.696	2.520	0.040	6.340	0.100
p2p-Gnutella31_40	2.260	2.116	0.048	5.096	0.104
p2p-Gnutella31_80	2.504	2.188	0.044	5.368	0.088
Amazon0302_05	6.936	10.276	0.188	32.136	1.740
Amazon0302_10	7.720	11.148	0.168	38.216	1.920
Amazon0302_20	6.480	10.088	0.152	34.404	1.668
Amazon0302_40	7.436	11.040	0.170	41.552	1.860
Amazon0302_80	8.376	11.892	0.172	45.224	2.084
soc-Epinions1_05	5.116	3.480	0.072	10.188	0.092
soc-Epinions1_10	10.708	7.684	0.092	21.540	0.112
soc-Epinions1_20	20.996	14.480	0.124	45.308	0.144
soc-Epinions1_40	43.012	31.372	0.204	97.144	0.204
soc-Epinions1_80	100.472	62.528	0.312	198.832	0.340
web-BerkStan_05	5.204	4.464	0.060	16.300	0.224
web-BerkStan_10	10.524	8.976	0.072	31.724	0.348
web-BerkStan_20	18.176	14.468	0.152	63.608	0.656
web-BerkStan_40	26.252	17.928	0.204	118.252	0.900
web-BerkStan_80	31.044	23.872	0.212	401.040	0.992
WikiTalk_05	71.068	97.160	0.984	159.068	2.120
WikiTalk_10	141.392	192.788	1.212	329.100	2.888
WikiTalk_20	282.890	386.008	1.772	945.708	4.100
WikiTalk_40	569.240	746.324	2.480	1533.260	6.436
WikiTalk_80	923.448	1122.870	3.364	2010.560	7.956
Amazon0601_05	871.088	790.916	1.424	2879.210	36.380
Amazon0601_10	1564.340	1417.060	2.524	4723.031	41.568
Amazon0601_20	2202.820	2118.520	1.920	4878.070	43.888
Amazon0601_40	2388.700	2068.550	3.756	7674.280	50.004
Amazon0601_80	2505.030	2395.961	4.276	7706.976	55.644
web-Google_05	2644.380	>2.5h	1.320	11914.512	6.792
web-Google_10	4767.340	>2.5h	1.756	>2.5h	10.420
web-Google_20	7160.680	>2.5h	3.344	>2.5h	16.476
web-Google_40	>2.5h	>2.5h	4.444	>2.5h	20.108
web-Google_80	>2.5h	>2.5h	4.892	>2.5h	20.528

The corresponding average running times are reported in the last two columns of Table 3. As above, our efficient algorithm Decr-LH is much faster than DSNCA-LH. We also observe that in most instances, maintaining a low-high order with our decremental algorithm Decr-LH incurs a relatively low overhead with respect to Decr. For some instances, such as product co-purchasing networks and web graphs (Amazon0302, Amazon0601, web-BerkStan and web-Google), however, the overhead of maintaining a low-high order is significantly higher. In our experiments this was due to the fact that a low-high order may change substantially (i.e., many vertices will be inserted into set X maintained by Algorithm FixLH), even if the dominator tree remains the same. (See Figure 2.)

References

- 1 S. Allesina and A. Bodini. Who dominates whom in the ecosystem? Energy flow bottlenecks and cascading extinctions. *Journal of Theoretical Biology*, 230(3):351–358, 2004.
- 2 S. Alstrup, D. Harel, P. W. Lauridsen, and M. Thorup. Dominators in Linear Time. *SIAM Journal on Computing*, 28(6):2117–32, 1999.
- 3 S. Alstrup and P. W. Lauridsen. A simple dynamic algorithm for maintaining a dominator tree. Technical Report 96-3, Department of Computer Science, University of Copenhagen, 1996.
- 4 M. E. Amyeen, W. K. Fuchs, I. Pomeranz, and V. Boppana. Fault Equivalence Identification Using Redundancy Information and Static and Dynamic Extraction. In *Proceedings of the 19th IEEE VLSI Test Symposium*, March 2001.
- 5 S. Baswana, K. Choudhary, and L. Roditty. Fault Tolerant Reachability for Directed Graphs. In Yoram Moses, editor, *Distributed Computing*, volume 9363 of *Lecture Notes in Computer Science*, pages 528–543. Springer Berlin Heidelberg, 2015. doi:10.1007/978-3-662-48653-5_35.
- 6 S. Baswana, K. Choudhary, and L. Roditty. Fault Tolerant Reachability Subgraph: Generic and Optimal. In *Proc. 48th ACM Symp. on Theory of Computing*, pages 509–518, 2016.
- 7 M. A. Bender, R. Cole, E. D. Demaine, M. Farach-Colton, and J. Zito. Two Simplified Algorithms for Maintaining Order in a List. In *Proc. 10th European Symposium on Algorithms*, pages 152–164, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. doi:10.1007/3-540-45749-6_17.
- 8 A. L. Buchsbaum, L. Georgiadis, H. Kaplan, A. Rogers, R. E. Tarjan, and J. R. Westbrook. Linear-Time Algorithms for Dominators and Other Path-Evaluation Problems. *SIAM Journal on Computing*, 38(4):1533–1573, 2008.
- 9 M. D. Carroll and B. G. Ryder. Incremental data flow analysis via dominator and attribute update. In *Proc. 15th ACM POPL*, pages 274–284, 1988.
- 10 S. Cicerone, D. Frigioni, U. Nanni, and F. Pugliese. A uniform approach to semi-dynamic problems on digraphs. *Theor. Comput. Sci.*, 203:69–90, August 1998.
- 11 R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, 13(4):451–490, 1991. doi:10.1145/115372.115320.
- 12 P. Dietz and D. Sleator. Two algorithms for maintaining order in a list. In *Proc. 19th ACM Symp. on Theory of Computing*, pages 365–372, 1987.
- 13 David Eppstein, Zvi Galil, and Giuseppe F. Italiano. Dynamic graph algorithms. In *Algorithms and Theory of Computation Handbook, 2nd Edition, Vol. 1*, pages 9.1–9.28. CRC Press, 2009.
- 14 W. Fraczak, L. Georgiadis, A. Miller, and R. E. Tarjan. Finding dominators via disjoint set union. *Journal of Discrete Algorithms*, 23:2–20, 2013. doi:10.1016/j.jda.2013.10.003.
- 15 H. N. Gabow. The Minset-Poset Approach to Representations of Graph Connectivity. *ACM Transactions on Algorithms*, 12(2):24:1–24:73, February 2016. doi:10.1145/2764909.
- 16 K. Gargi. A Sparse Algorithm for Predicated Global Value Numbering. *SIGPLAN Not.*, 37(5):45–56, May 2002. doi:10.1145/543552.512536.

- 17 L. Georgiadis. Testing 2-vertex connectivity and computing pairs of vertex-disjoint s - t paths in digraphs. In *Proc. 37th Int'l. Coll. on Automata, Languages, and Programming*, pages 738–749, 2010.
- 18 L. Georgiadis. Approximating the Smallest 2-Vertex Connected Spanning Subgraph of a Directed Graph. In *Proc. 19th European Symposium on Algorithms*, pages 13–24, 2011.
- 19 L. Georgiadis, K. Giannis, A. Karanasiou, and L. Laura. Incremental Low-High Orders of Directed Graphs and Applications. In Costas S. Iliopoulos, Solon P. Pissis, Simon J. Puglisi, and Rajeev Raman, editors, *16th International Symposium on Experimental Algorithms (SEA 2017)*, volume 75 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:21, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.SEA.2017.27.
- 20 L. Georgiadis, T. D. Hansen, G. F. Italiano, S. Krinninger, and N. Parotsidis. Decremental Data Structures for Connectivity and Dominators in Directed Graphs. In *ICALP*, pages 42:1–42:15, 2017.
- 21 L. Georgiadis, G. F. Italiano, L. Laura, and N. Parotsidis. 2-Edge Connectivity in Directed Graphs. In *Proc. 26th ACM-SIAM Symp. on Discrete Algorithms*, pages 1988–2005, 2015.
- 22 L. Georgiadis, G. F. Italiano, L. Laura, and N. Parotsidis. 2-Vertex Connectivity in Directed Graphs. In *Proc. 42nd Int'l. Coll. on Automata, Languages, and Programming*, pages 605–616, 2015.
- 23 L. Georgiadis, G. F. Italiano, L. Laura, and F. Santaroni. An Experimental Study of Dynamic Dominators. In *Proc. 20th European Symposium on Algorithms*, pages 491–502, 2012. Full version: *CoRR*, arXiv:1604.02711.
- 24 L. Georgiadis, G. F. Italiano, and N. Parotsidis. Incremental 2-Edge-Connectivity in Directed Graphs. In *ICALP*, pages 49:1–49:15, 2016.
- 25 L. Georgiadis and R. E. Tarjan. Dominator Tree Certification and Divergent Spanning Trees. *ACM Transactions on Algorithms*, 12(1):11:1–11:42, November 2015. doi:10.1145/2764913.
- 26 L. Georgiadis and R. E. Tarjan. Addendum to “Dominator Tree Certification and Divergent Spanning Trees”. *ACM Transactions on Algorithms*, 12(4):56:1–56:3, August 2016. doi:10.1145/2928271.
- 27 L. Georgiadis, R. E. Tarjan, and R. F. Werneck. Finding Dominators in Practice. *Journal of Graph Algorithms and Applications (JGAA)*, 10(1):69–94, 2006.
- 28 M. Gomez-Rodriguez and B. Schölkopf. Influence Maximization in Continuous Time Diffusion Networks. In *29th International Conference on Machine Learning (ICML)*, 2012.
- 29 Andreas D.M. Gunawan, Bhaskar DasGupta, and Louxin Zhang. A decomposition theorem and two algorithms for reticulation-visible networks. *Information and Computation*, 252:161–175, 2017. doi:10.1016/j.ic.2016.11.001.
- 30 M. S. Hecht and J. D. Ullman. Characterizations of Reducible Flow Graphs. *Journal of the ACM*, 21(3):367–375, 1974.
- 31 M. Henzinger, S. Krinninger, and V. Loitzenbauer. Finding 2-edge and 2-vertex strongly connected components in quadratic time. In *Proc. 42nd Int'l. Coll. on Automata, Languages, and Programming*, pages 713–724, 2015.
- 32 G. F. Italiano, L. Laura, and F. Santaroni. Finding strong bridges and strong articulation points in linear time. *Theoretical Computer Science*, 447:74–84, 2012. doi:10.1016/j.tcs.2011.11.011.
- 33 R. Jaber. Computing the 2-blocks of directed graphs. *RAIRO-Theor. Inf. Appl.*, 49(2):93–119, 2015. doi:10.1051/ita/2015001.
- 34 R. Jaber. On computing the 2-vertex-connected components of directed graphs. *Discrete Applied Mathematics*, 204:164–172, 2016. doi:10.1016/j.dam.2015.10.001.
- 35 Jérôme Kunegis. KONECT: the Koblenz network collection. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013, Companion Volume*, pages 1343–1350, 2013. doi:10.1145/2487788.2488173.

- 36 T. Lengauer and R. E. Tarjan. A Fast Algorithm for Finding Dominators in a Flowgraph. *ACM Transactions on Programming Languages and Systems*, 1(1):121–41, 1979.
- 37 J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. <http://snap.stanford.edu/data>, June 2014.
- 38 E. K. Maxwell, G. Back, and N. Ramakrishnan. Diagnosing memory leaks using graph mining on heap dumps. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 115–124, 2010.
- 39 R. M. McConnell, K. Mehlhorn, S. Näher, and P. Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, 2011.
- 40 L. Quesada, P. Van Roy, Y. Deville, and R. Collet. Using Dominators for Solving Constrained Path Problems. In *Proc. 8th International Conference on Practical Aspects of Declarative Languages*, pages 73–87, 2006.
- 41 G. Ramalingam and T. Reps. An incremental algorithm for maintaining the dominator tree of a reducible flowgraph. In *POPL*, pages 287–296, 1994.
- 42 V. C. Sreedhar, G. R. Gao, and Y. Lee. Incremental computation of dominator trees. *ACM Transactions on Programming Languages and Systems*, 19:239–252, 1997.
- 43 R. E. Tarjan. Finding Dominators in Directed Graphs. *SIAM Journal on Computing*, 3(1):62–89, 1974.
- 44 R. E. Tarjan. Testing Flow Graph Reducibility. *J. Comput. Syst. Sci.*, 9(3):355–365, 1974. doi:10.1016/S0022-0000(74)80049-8.
- 45 T. Tholey. Linear time algorithms for two disjoint paths problems on directed acyclic graphs. *Theoretical Computer Science*, 465:35–48, 2012. doi:10.1016/j.tcs.2012.09.025.
- 46 J. Zhao and S. Zdancewic. Mechanized Verification of Computing Dominators for Formalizing Compilers. In *Proc. 2nd International Conference on Certified Programs and Proofs*, pages 27–42. Springer, 2012. doi:10.1007/978-3-642-35308-6_6.

On the Hardness and Inapproximability of Recognizing Wheeler Graphs

Daniel Gibney 

Department of Computer Science, University of Central Florida, Orlando FL, USA

<http://cs.ucf.edu/~dgibney/>

Daniel.Gibney@ucf.edu

Sharma V. Thankachan

Department of Computer Science, University of Central Florida, Orlando FL, USA

<http://www.cs.ucf.edu/~sharma/>

sharma.thankachan@ucf.edu

Abstract

In recent years several *compressed indexes* based on variants of the Burrows-Wheeler transformation have been introduced. Some of these are used to index structures far more complex than a single string, as was originally done with the FM-index [Ferragina and Manzini, J. ACM 2005]. As such, there has been an increasing effort to better understand under which conditions such an indexing scheme is possible. This has led to the introduction of Wheeler graphs [Gagie et al., Theor. Comput. Sci., 2017]. Gagie et al. showed that de Bruijn graphs, generalized compressed suffix arrays, and several other BWT related structures can be represented as Wheeler graphs, and that Wheeler graphs can be indexed in a way which is space efficient. Hence, being able to recognize whether a given graph is a Wheeler graph, or being able to approximate a given graph by a Wheeler graph, could have numerous applications in indexing. Here we resolve the open question of whether there exists an efficient algorithm for recognizing if a given graph is a Wheeler graph. We present:

- The problem of recognizing whether a given graph $G = (V, E)$ is a Wheeler graph is NP-complete for any edge label alphabet of size $\sigma \geq 2$, even when G is a DAG. This holds even on a restricted, subset of graphs called d -NFA's for $d \geq 5$. This is in contrast to recent results demonstrating the problem can be solved in polynomial time for d -NFA's where $d \leq 2$. We also show the recognition problem can be solved in linear time for $\sigma = 1$;
- There exists an $2^{e \log \sigma + O(n+e)}$ time exact algorithm where $n = |V|$ and $e = |E|$. This algorithm relies on graph isomorphism being computable in strictly sub-exponential time;
- We define an optimization variant of the problem called Wheeler Graph Violation, abbreviated WGV, where the aim is to remove the minimum number of edges in order to obtain a Wheeler graph. We show WGV is APX-hard, even when G is a DAG, implying there exists a constant $C \geq 1$ for which there is no C -approximation algorithm (unless $P = NP$). Also, conditioned on the Unique Games Conjecture, for all $C \geq 1$, it is NP-hard to find a C -approximation;
- We define the Wheeler Subgraph problem, abbreviated WS, where the aim is to find the largest subgraph which is a Wheeler Graph (the dual of the WGV). In contrast to WGV, we prove that the WS problem is in APX for $\sigma = O(1)$;

The above findings suggest that most problems under this theme are computationally difficult. However, we identify a class of graphs for which the recognition problem is polynomial time solvable, raising the open question of which parameters determine this problem's difficulty.

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

Keywords and phrases Burrows-Wheeler transform, string algorithms, suffix trees, NP-completeness

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.51

Related Version A full version of the paper is available at <https://arxiv.org/abs/1902.01960>.

Funding was received from the U.S. National Science Foundation (NSF) under CCF-1703489, European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 690941.

Acknowledgements We would like to thank T. Gagie and N. Prezza for their valuable feedback.



© Daniel Gibney and Sharma V. Thankachan;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 51; pp. 51:1–51:16



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Within the last two decades, there has been the development of Burrows Wheeler Transform (BWT) [7] based indices for compressing a diverse collection of data structures. This list includes labeled trees [31], certain classes of graphs [14, 29], and sets of multiple strings [16, 26]. This has motivated the search for a set of general conditions under which a structure can be indexed by a BWT based index, and consequently the introduction of Wheeler graphs. A Wheeler graph is a directed graph with edge labels which satisfies two simple axioms related to the ordering of its vertices. They were introduced by Gagie et al. [17] (also see [2]). Although not general enough to encompass all BWT-based structures (e.g., [18]), the authors demonstrated that Wheeler graphs offer a unified way of modeling several BWT based data structures such as representations of de Bruijn graphs [6, 10], generalized compressed suffix arrays [31], multistring BWTs [27], XBWTs [14], wavelet matrices [9], and certain types of finite automaton [1, 5, 24]. They also showed that there exists an encoding of a Wheeler graph $G = (V, E)$ which requires only $2(e + n) + e \log \sigma + \sigma \log e + o(n + e \log \sigma)$ bits where σ is the size of the edge label alphabet, $e = |E|$, and $n = |V|$. This encoding allows for the efficient traversal of multiple edges while processing characters in a string, using an algorithm similar to the backward search in the FM-index [15]. Since their introduction Wheeler graphs have been further developed using techniques such as tunneling by Alanko et al. [2]. Also, ideas closely coupled to Wheeler graphs have shown up in the recent work by Equi et al. for exact pattern matching on graphs [12, 13]. However, not all directed edge labeled graphs are Wheeler graphs, and thus not all directed edge labeled graphs can have such techniques applied to them. The authors of [17] posed the question of *how to reasonably recognize whether a given graph is a Wheeler graph*.

The question is of both theoretical and practical value, as it might be the first step before attempting to apply some compression scheme to a given graph. For example, one could use the existence of a *Wheeler subgraph* to encode a graph. To do so, you could maintain an encoding of the subgraph using the framework presented in [17] in addition to an adjacency list of the edges not included in the encoding. Depending on the size of the subgraph, such an encoding might provide a large space savings at the cost of a modest time trade-off while traversing the graph. This concept also motivates the portion of the paper where we look at *optimization versions* of this problem that seek subgraphs of the given graph which are Wheeler graphs. Unfortunately for practitioners of such a method, this problem turns out to be computationally difficult as well. As a positive result, we show that the problem of finding a maximal Wheeler subgraph admits a polynomial time algorithm which has solution size within some constant factor of optimal for constant alphabet size. We also show that the problem of recognizing Wheeler graphs is similar to that of identifying the queue number of a graph, indicating a class of graphs where the problem becomes computationally tractable.

1.1 Wheeler Graphs

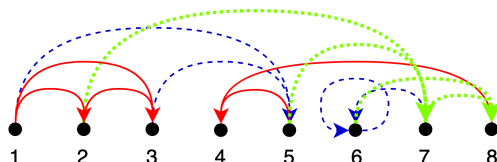
The notation (u, v, k) is used for the directed edge from u to v with label k . We will assume the usual ordering on the edge labels which come from the alphabet $\{1, 2, \dots, \sigma\}$.

► **Definition 1.** *A Wheeler graph is a directed graph with edge labels where there exists an ordering π on the vertices such that for any two edges (u, v, k) and (u', v', k') :*

- (i) $k < k' \implies v <_{\pi} v'$;
- (ii) $(k = k') \wedge (u <_{\pi} u') \implies v \leq_{\pi} v'$.

In addition, vertices with in-degree zero must be placed first in the ordering.

We consider an ordering of the vertices of the graph a *proper* ordering if it satisfies the axioms of the Wheeler graph definition. See Figure 1 for an illustration. One critical property of Wheeler graphs is called *path coherence*. This property is characterized by the fact that if you start at any consecutive range of vertices under the proper ordering π , and traverse the graph following edge labels matching the characters in a string P , then when finished processing P the vertices ended on will form a consecutive range. This property is key to allowing the efficient traversal of multiple edges simultaneously, as well as achieving a compressed representation of the graph.



■ **Figure 1** An example of a Wheeler graph with $\sigma = 3$. The ordering on the edge labels is: red (solid) < blue (long-dash) < green (short-dash).

The following list of properties for a Wheeler graph can be deduced from Definition 1.

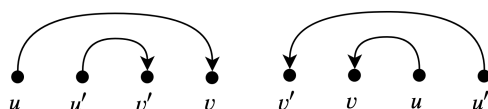
- ▶ **Property 1.** *All edges inbound to a vertex v have the same edge label.*
- ▶ **Property 2.** *In a proper ordering all vertices with same inbound edge label are ordered consecutively.*
- ▶ **Property 3.** *It is possible for a vertex to have multiple outbound edges with the same label. It is also possible for a vertex to have more than σ inbound or outbound edges.*
- ▶ **Property 4.** *We call two edges (u, v, k) and (u', v', k) with the same label a rainbow if $u < u'$ and $v' < v$. No rainbows can exist in a proper ordering (see Figure 2).*
- ▶ **Property 5.** *For $\sigma = 1$, ignoring self-loops, any proper ordering is also a topological ordering. The vertices with self-loops must be placed last in the ordering otherwise a rainbow is created.*

1.2 Problem Definitions

The first question we wish to answer is given a directed graph with edge labels, does a proper ordering π exist? We define this problem formally as the following.

▶ **Problem 1 (Wheeler Graph Recognition).** *Given a directed edge labeled graph $G = (V, E)$, answer “YES” if G is a Wheeler graph and “NO” otherwise.*

Although we do not demand it here, ideally, a solution to the above problem would also return a proper ordering.



■ **Figure 2** In a proper ordering all of the above configurations cannot occur.

Motivated by the compression of general graphs (which are not necessarily Wheeler graphs), we next define two optimization versions of Problem 1 where we seek to find Wheeler subgraphs.

► **Problem 2** (Wheeler Graph Violation (WGV)). *Given a directed edge labeled graph $G = (V, E)$ identify the smallest $E' \subseteq E$ such that $G' = (V, E \setminus E')$ is a Wheeler graph.*

We also consider the dual of this problem.

► **Problem 3** (Wheeler Subgraph (WS)). *Given a directed edge labeled graph $G = (V, E)$ identify the largest $E'' \subseteq E$ such that $G'' = (V, E'')$ is a Wheeler graph.*

1.3 Our Contribution

- In Section 2 we show that the problem of recognizing whether a given graph is a Wheeler graph is NP-complete even for an edge alphabet of size $\sigma = 2$. The result holds even when the input is a directed acyclic graph (DAG) and when the number of edges leaving a vertex with the same label is at most five. In the full version of this paper [19] we show that for $\sigma = 1$ the recognition problem can be reduced to that of determining if a DAG has queue number one which can be solved in linear time [22].
- In Section 3 we provide an exponential time algorithm which solves the recognition problem on a graph $G = (V, E)$ in time $2^{O(n+e \log \sigma)}$ where $n = |V|$ and $e = |E|$. It uses the idea of enumerating through all possible encodings (of bounded size) of Wheeler graphs, and the fact that we can test whether there exists an isomorphism between two undirected graphs in sub-exponential time. This technique also gives us exact algorithms for the optimization problems presented in this paper which run in time $2^{O(n+e \log \sigma)}$.
- Section 4 examines the optimization versions of this problem called Wheeler Graph Violation (WGV) and Wheeler Subgraph (WS). We show via a reduction of the Minimum Feedback Arc Set problem that the Wheeler Graph Violation problem is APX-hard, and assuming the Unique Games Conjecture, cannot be approximated within a constant factor. This holds even when the graph is a DAG. On the other hand, we show that the Wheeler Subgraph problem is in the complexity class APX for $\sigma = O(1)$. We do so by providing a poly-time algorithm whose solution size is $\Omega(1/\sigma)$ times the optimal value.
- Using PQ-trees and ideas similar to those used in detecting if the queue number of a DAG is one, we demonstrate a class of graphs where Wheeler graph recognition can be done in linear time (see full version [19]).

2 NP-completeness of Wheeler Graph Recognition

► **Theorem 2.** *The Wheeler Graph Recognition Problem is NP-complete for any $\sigma \geq 2$.*

We first show a simple reduction from the Betweenness problem to Wheeler Graph Recognition. Although straightforward, it requires graphs with either $O(n)$ sources or $O(n)$ edges with the same label leaving a single vertex. In Section 2.3, by expanding on the techniques used in the first reduction we show that even if these quantities are limited to at most five the recognition problem remains NP-complete.

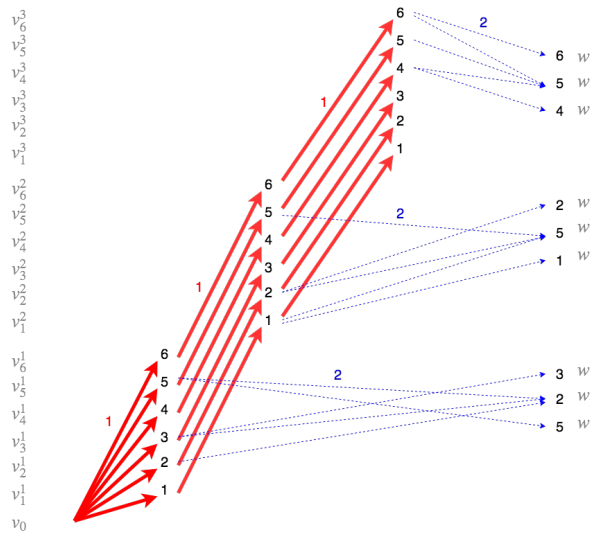
2.1 The Betweenness Problem

The Betweenness Problem was established as NP-complete by Opatrný in 1979 [30]. Like our problem, it deals with finding a total ordering on a set of elements subject to some constraints. The input to the Betweenness problem is a list of distinct elements $T = t_1, \dots, t_n$ and a collection of $k < n^3$ ordered triples of $(t_1^1, t_2^1, t_3^1), (t_1^2, t_2^2, t_3^2), \dots, (t_1^k, t_2^k, t_3^k)$ where every element

in a triple is in T . The list T should be placed into a total ordering with the property that for each of the given triples the middle item in the triple appears somewhere between the other two items. The items of each triple are not required to be consecutive in the total ordering. The decision problem is determining if such an ordering is possible.

As an example, consider the input $T = 1, 2, 3, 4, 5, 6$ and the triples: $(5, 2, 3)$, $(1, 5, 2)$, $(4, 5, 6)$, $(4, 6, 2)$. A total ordering which satisfies the given triples is $1, 4, 5, 6, 2, 3$. An ordering which does not satisfy the given triples is $1, 2, 3, 4, 5, 6$ since it violates the triples $(5, 2, 3)$, $(1, 5, 2)$, and $(4, 6, 2)$.

2.2 Reduction from Betweenness to Wheeler Graph Recognition



■ **Figure 3** An example of the reduction with input list $1, 2, 3, 4, 5, 6$ and triples $(5, 2, 3), (1, 5, 2), (4, 5, 6)$.

Suppose we are given as input to the Betweenness Problem the list t_1, t_2, \dots, t_n and triples $(t_1^1, t_2^1, t_3^1), (t_1^2, t_2^2, t_3^2), \dots, (t_1^k, t_2^k, t_3^k)$. We construct a DAG of size $O(nk)$ as follows:

- Create a source vertex v_0 and vertices v_i^j for $1 \leq i \leq n$ and $1 \leq j \leq k$.
- For each triple (t_1^j, t_2^j, t_3^j) create a vertex for each element of the triple, we call them w_1^j, w_2^j , and w_3^j respectively.
- Create the edges $(v_0, v_i^1, 1)$ and edges $(v_i^j, v_i^{j+1}, 1)$ for $1 \leq i \leq n, 1 \leq j \leq k - 1$.
- Create the following edges:
 - $(v_i^j, w_1^j, 2)$ if $t_i = t_1^j$
 - $(v_i^j, w_2^j, 2)$ if $t_i = t_2^j$
 - $(v_i^j, w_3^j, 2)$ if $t_i = t_3^j$
 - $(v_i^j, w_2^j, 2)$ if $t_i = t_1^j$
 - $(v_i^j, w_1^j, 2)$ if $t_i = t_3^j$

► **Lemma 3.** *An instance of the Betweenness problem has an ordering satisfying all of the constraints iff the graph constructed as above is a Wheeler graph.*

Proof Sketch. The intuition is that the vertices with inbound edge label 1 represent the permutation of the elements in T . The edges labeled 1 force the permutation to be duplicated k times, once for each constraint. The vertices with the inbound edge label 2 represent the

elements in each triple. The edges with label 2 enforce that the only valid orderings of the vertices representing elements in T are orderings that satisfy the Betweenness constraints. This is enforced by having no edges labeled 2 which are crossing in the figure. The detailed proof can be found in the full version of this paper [19]. ◀

Theorem 2 follows from Lemma 3. The fact that being a Wheeler graph implies (arched) level planarity with respect to each edge label is the key to the reduction.

The Wheeler graph recognition problem can be solved in linear time for an alphabet of size one. This follows from relating the notion of queue number to Wheeler graphs, and previous results which give a linear time algorithm for finding a one-queue DAG [21, 22, 23]. This also gives an upper bound on the number of edges which can be in a Wheeler graph [11]. The proof can be found in the full version of this paper [19].

► **Theorem 4.** *The Wheeler graph recognition problem can be solved in linear time for an edge alphabet of size $\sigma = 1$.*

► **Theorem 5.** *For $\sigma = 1$, the number of edges in a Wheeler graph is $O(n)$.*

2.3 NP-completeness of Wheeler Graph Recognition on d -NFA's

Now we restrict the number of edges with the same label that can leave a single vertex. We adopt the terminology used by Alanko, Policriti and Prezza and consider the problem of recognizing whether a d -NFA is also a Wheeler graph [3]. A d -NFA is defined as follows:

► **Definition 6.** *A d -NFA G is an NFA where the number of edges with the same character leaving a vertex is at most d . We refer to the value d as the non-determinism of G .*

We emphasize that here an NFA contains a single start state, from which we assume each vertex is reachable. The results in this section are in contrast to the recent work of Alanko, Policriti and Prezza who showed that it can be recognized in polynomial time whether a 2-NFA is a Wheeler graph [3]. Their result coupled with the observation that the reduction in Section 2 requires a $n^{\Theta(1)}$ -NFA suggests an interesting question about what role non-determinism plays in the tractability of Wheeler graph recognition. To this end, we prove Theorem 7.

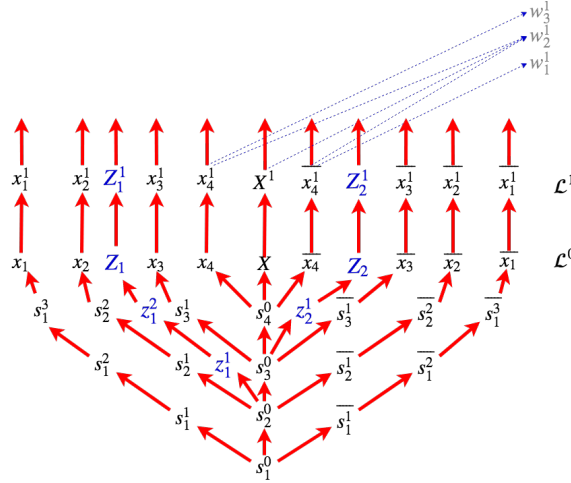
► **Theorem 7.** *The Wheeler Graph Recognition Problem is NP-complete for d -NFA's, $d \geq 5$.*

The strategy of the proof is to reduce the NP-complete problem 4-NAESAT to Wheeler Graph Recognition. In 4-NAESAT each clause is of length 4, and an expression is satisfiable iff there exists a truth assignment such that each clause contains both a true literal and a false literal. We start with 4-NAESAT, rather than 3-NAESAT, to obtain a 3-NAESAT instance with the special property highlighted by Lemma 8.

► **Lemma 8.** *An instance ϕ of 4-NAESAT can be reduced in poly-time to an instance ϕ' of 3-NAESAT where a variable occurring in the middle of a clause appears at most twice in ϕ' .*

Proof. Convert the 4-NAESAT instance ϕ to a 3-NAESAT instance ϕ' by converting each clause (a_k, b_k, c_k, d_k) into the clauses (a_k, w_k, b_k) and $(c_k, \overline{w_k}, d_k)$. Both clauses have a satisfying not-all-equal assignment iff it is not the case that $a_k = b_k = c_k = d_k$. We note that the variable used in the middle of the clauses, w_k , is used only twice in ϕ' . ◀

For convenience, we define a case of 3-NAESAT where each variable occurring in the middle occurs at most twice, we call this 3-NAESAT*. We next describe the construction of a one source DAG from an instance of 3-NAESAT*.



■ **Figure 4** Vertex Z_1 and Z_2 could be for clauses (x_1, x_2, x_3) , (x_2, \bar{x}_3, x_4) . Each “betweenness” constraint adds a layer. (x_4, X, \bar{x}_4) constraint shown.

Suppose we are given an instance ϕ of 3-NAESAT* with variables x_1, x_2, \dots, x_n and the clauses (a_k, b_k, c_k) where we assume a_k, b_k, c_k can represent either a Boolean variable or its negation. We create a single source DAG G based on ϕ . The first step creates a *menorah like* structure which allows for the vertices representing x_i and \bar{x}_i to swap places in G , but otherwise fixes the positions of the vertices. We begin by adding the vertices which represent our variables, $x_1, \dots, x_n, X, \bar{x}_1, \dots, \bar{x}_n$; (the role of X will become clear). Next, we add the structure to constrain their possible positions.

Add to G the vertices:

- s_1^0, \dots, s_n^0 ;
- For $1 \leq i \leq n-1, 1 \leq j \leq n-i$: s_i^j and \bar{s}_i^j ;

Add to G the red edges:

- $(s_1^0, s_2^0, 1), \dots, (s_n^0, X, 1)$;
- For $1 \leq i \leq n-1, 2 \leq j \leq n-i$: $(s_i^{j-1}, s_i^j, 1)$ and $(\bar{s}_i^{j-1}, \bar{s}_i^j, 1)$;
- For $1 \leq i \leq n-1$: $(s_i^0, s_i^1, 1)$ and $(\bar{s}_i^0, \bar{s}_i^1, 1)$;
- For $1 \leq i \leq n$: $(s_i^{n-i}, x_i, 1)$ and $(\bar{s}_i^{n-i}, \bar{x}_i, 1)$;

For clause k , denoted (a_k, b_k, c_k) , we add a vertex Z_k . Suppose the middle variable of the clause, b_k , is x_h (positive or negated), then we add the vertices z_k^j for $1 \leq j \leq n-h$, and red edges $(s_h^0, z_k^1, 1), (z_k^1, z_k^2, 1) \dots (z_k^{n-h}, Z_k, 1)$.

Now we wish add a set of *betweenness* type constraints on any proper ordering given of the vertices $\mathcal{L}^0 = \{x_1, \dots, X, \bar{x}_1, \dots, \bar{x}_n, Z_1, Z_2, \dots\}$. Given a constraint (y_1, y_2, y_3) we insist y_2 be between y_1 and y_3 in the ordering. This can be enforced by adding a layer of new vertices $\mathcal{L}^1 = \{x_1^1, \dots, X^1, \bar{x}_1^1, \dots, \bar{x}_n^1, Z_1^1, Z_2^1, \dots\}$ with red edges labeled 1 from vertices in layer \mathcal{L}^0 to their corresponding vertices in \mathcal{L}^1 . We use the same gadget that was used in Section 2. Consider adding a betweenness on the vertices y_1, y_2, y_3 in \mathcal{L}^0 . Add the vertices w_1^1, w_2^1 , and w_3^1 and the blue edges $(y_1^1, w_1^1, 2), (y_2^1, w_2^1, 2), (y_3^1, w_3^1, 2), (y_1^1, w_2^1, 2)$ and $(y_3^1, w_2^1, 2)$. Additional betweenness constraints can be similarly enforced by adding a new layer on top of \mathcal{L}^1 with a new gadget. Add the betweenness constraints (x_i, X, \bar{x}_i) for $1 \leq i \leq n$ fixing X , and betweenness constraints (a_k, Z_k, b_k) and (c_k, X, Z_k) for every clause (a_k, b_k, c_k) .

■ **Table 1** Possible relative orderings of a_k, b_k, c_k, Z_k, X subject to (a_k, Z_k, b_k) and (c_k, X, Z_k) .

Possible Orderings (a_k has variable x_j and c_k has variable x_h)		
$a_k b_k c_k$	$j < h$	$h < j$
FFT	$c_k \dots X \dots b_k, Z_k \dots a_k$	$c_k \dots X \dots b_k, Z_k \dots a_k$
FTF	$b_k, Z_k \dots X \dots c_k \dots a_k$	$b_k, Z_k \dots X \dots a_k \dots c_k$
TFF	$a_k \dots \bar{b}_k, Z_k \dots X \dots b_k \dots c_k$	$a_k \dots \bar{b}_k, Z_k \dots X \dots b_k \dots c_k$
FTT	$c_k \dots b_k \dots X \dots \bar{b}_k, Z_k \dots a_k$	$c_k \dots b_k \dots X \dots \bar{b}_k, Z_k \dots a_k$
TFT	$a_k \dots c_k \dots X \dots Z_k, b_k$	$c_k \dots a_k \dots X \dots Z_k, b_k$
TTF	$a_k \dots Z_k, b_k \dots X \dots c_k$	$a_k \dots Z_k, b_k \dots X \dots c_k$

■ **Table 2** Orderings implied by all-equal assignment are not possible while satisfying constraints.

(Not) Possible Orderings (a_k has variable x_j and c_k has variable x_h)		
$a_k b_k c_k$	$j < h$	$h < j$
TTT	$a_k \dots b_k \dots c_k \dots X$	$c_k \dots b_k \dots a_k \dots X$
FFF	$X \dots c_k \dots b_k \dots a_k$	$X \dots a_k \dots b_k \dots c_k$

Before proving the correctness of the reduction, we make the observation that because any variable occurring in the middle of a clause occurs as most **twice** in the whole expression, the maximum number of edges leaving a vertex s_i^0 is bounded by $3 + 2 = 5$. All of the other vertices have at most three edges with the same label leaving them.

► **Lemma 9.** *The leveled graph G constructed as above from an instance ϕ' of 3-NAESAT* is a Wheeler graph iff ϕ' is satisfiable.*

Proof. Given a truth assignment that satisfies the 3-NAESAT* instance ϕ' , put the vertices in \mathcal{L}^0 whose variables are assigned the value T on the left side of X in Figure 4, and the vertices whose variables are assigned F on the right side of X . For example, if $x_1 = T, x_2 = F$, the two left-most vertex on level \mathcal{L}^0 would be x_1 followed by \bar{x}_2 . Now, for clause (a_k, b_k, c_k) we have the possible not-all-equal truth assignments and relative orderings of \mathcal{L}^0 which satisfy the Wheeler graph axioms in Table 1. This shows that a Wheeler graph ordering of the vertices is possible by placing Z_k 's correctly given the truth assignment.

In the other direction, if G is a Wheeler graph then the ordering of the menorah structure is fixed with the exception of z_i^j vertices and the ordering duplicated across layers $\mathcal{L}^0, \mathcal{L}^1, \dots$. We will show the ordering given to \mathcal{L}^0 must have every clause getting a not-all-equal assignment. Suppose to the contrary that \mathcal{L}^0 was given an ordering where either a_k, b_k, c_k are all on the left(true) or the right side(false) of X . Then we have the options in Table 2.

In all cases listed in Table 2, placing Z_k between a_k and b_k violates the constraint (c_k, X, Z_k) , implying we violate a Wheeler graph constraint as well, a contradiction. Hence, if G is a Wheeler graph, a valid ordering for \mathcal{L}^0 implies a valid truth assignment for ϕ' . ◀

This leaves open the complexity of the recognition problem for 3-NFA's and 4-NFA's.

3 An Exponential Time Algorithm

We can apply the encoding introduced by Gagie et al. [17] to develop exponential time algorithms to solve all of the problems presented in this paper. The idea is to enumerate over all possible encodings of Wheeler graphs with the proper number of vertices, edges, and labels, checking whether the encoding is isomorphic with the given graph. This idea exploits

the fact that having such a space efficient encoding also implies have a limited search space of Wheeler graphs. The graph isomorphism can be checked efficiently enough to maintain the desired time complexity. The results are summarized in the following two theorems.

► **Theorem 10.** *Recognizing whether $G = (V, E)$ is a Wheeler graph can be done in time $2^{e \log \sigma + O(n+e)}$, where $n = |V|$, $e = |E|$, and σ is the size of the edge label alphabet.*

Before describing the algorithm that proves Theorem 10 we need to describe the encoding of a Wheeler graph given in [17]. A Wheeler graph can be completely specified by three bit vectors. Two bit vectors I and O both of length $e + n$ and a bit vector L of length $e \log \sigma$. We assume that the vertices of the Wheeler graph G are listed in a proper ordering $x_1 <_{\pi} x_2 <_{\pi} \dots <_{\pi} x_n$. The array I is of the form $0^{\ell_1} 1 0^{\ell_2} 1 \dots 0^{\ell_n} 1$ and O is of the form $0^{k_1} 1 0^{k_2} 1 \dots 0^{k_n} 1$. Here ℓ_i is the out-degree of x_i whereas k_i is the in-degree of x_i . The array L indicates which of the e character symbols are assigned to each edge. Specifically, the i^{th} character in L gives us the label of the edge corresponding to the i^{th} zero in O . In [17] an additional C array is added, and these arrays are equipped with additional rank and select structures to allow for efficient traversal as is done in the FM-index [15]. For our purposes, however, the arrays O , I , and L are adequate.

The outline of the algorithm is given below as Algorithm 1. It essentially enumerates all bit vectors of a given length, checks whether or not the bit vector encodes a valid Wheeler graph, and if so then checks whether the encoding matches our given graph G . Let S represent the set of all possible encodings we wish to check. Note that $|S| \leq 2^{2(e+n)+e \log \sigma}$.

■ **Algorithm 1** IdentifyWheelerGraph(G).

```

for all  $(O, I, L) \in S$  do
  if  $(O, I, L)$  defines a valid wheeler graph  $G'$  then
    convert  $G$  to undirected graph  $\alpha(G)$ 
    convert  $G'$  to undirected graph  $\alpha(G')$ 
    if  $\alpha(G)$  and  $\alpha(G')$  are isomorphic then
      return "Wheeler Graph"
    end if
  end if
end for
return "Not a Wheeler Graph"

```

The Wheeler graph corresponding to the encoding can be extracted by working from right to left reading the array I . For each zero in I , we know which symbol should be on the inbound edge going into the corresponding vertex. We only need to decide where the edge's tail was. Let k be the edge label and j be the index of the label k in L which is furthest to the right in L and yet to be used. If no such j exists we reject the encoding. When assigning the tail for an edge, take as the tail the vertex x_i where $i = \text{rank}_1(O, \text{select}_0(O, j))$. We call the graph constructed in this way G' .

We now wish to check whether G' and G are the same graphs only with a reordering of the vertices, that is, G' is the result of applying an isomorphism to G . Unlike the typical isomorphism for labeled graphs, where a bijection between the symbols on the edge alphabet is all that is required, here we wish for the adjacency and the label on the edge to be preserved in the mapping between G and G' . Specifically, we wish to know if there exists a bijective

function $f : V(G) \rightarrow V(G')$, such that if $u, v \in V(G)$ are adjacent via an edge (u, v, k) with label k in G , then $f(u)$ and $f(v)$ are also adjacent via an edge $(f(u), f(v), k)$ with label k in G' . Using ideas similar to those presented by Miller in [28], this problem can be reduced in polynomial time to checking whether two undirected graphs are isomorphic.

► **Lemma 11.** *Checking whether the direct edge labeled graph G' is edge label preserving isomorphic to G can be reduced in polynomial time to checking if two undirected graphs are isomorphic.*

Proof. See full version [19]. ◀

The final step in this algorithm is to check whether $\alpha(G)$ and $\alpha(G')$ are isomorphic. Using well established techniques this can be done in time $2^{\sqrt{n'}+O(1)}$ where n' is the number of vertices in $\alpha(G)$ [4]. The total time complexity of Algorithm 1 is the number of bit strings tested, multiplied by the time it takes to (1) validate whether the bit string encodes a Wheeler graph G' and decode it, (2) convert G and G' to undirected graphs $\alpha(G)$ and $\alpha(G')$, and (3) test whether $\alpha(G)$ and $\alpha(G')$ are isomorphic. This yields an overall time complexity of $|S|n^{O(1)}2^{\sqrt{n+2e(\sigma+1)+O(1)}}$, i.e., $2^{e \log \sigma + O(n+e)}$ for Algorithm 1.

4 Optimization Variants to Wheeler Graph Recognition

4.1 The Wheeler Graph Violation Problem is APX-hard

In this section, we show that obtaining an approximate solution to the WGV problem that comes within a constant factor of the optimal solution is NP-hard. We do this through a reduction that shows that WGV is at least as hard as solving the Minimum Feedback Arc Set problem (FAS). FAS in its original formulation is phrased in terms of a directed graph where the objective is to find the minimum number of edges which need to be removed in order to make the directed graph a DAG. A slightly different formulation proves more useful for us. Letting $F_\pi = \{(v_i, v_j) \in E \mid \pi(v_i) > \pi(v_j)\}$ we have the following:

► **Lemma 12** (Younger [32]). *Determining a minimum feedback arc set for $G = (V, E)$ is equivalent to finding an ordering π on V for which $|F_\pi|$ is minimized.*

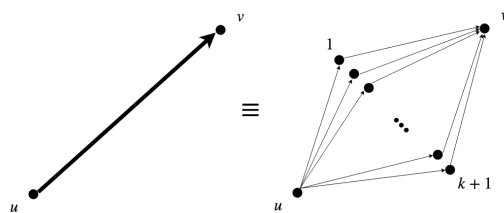
From this, we can present the equivalent formulation of FAS.

► **Definition 13** (Minimum Feedback Arc Set (FAS)). *The input is a list $T = t_1 t_2 \dots t_n$ of n numbers and a set of k inequalities of the form $t_i < t_j$. This task is to compute an ordering π on T so that the number of inequalities violated is minimized.*

Interestingly, we could not have used FAS for proving that the Wheeler graph recognition problem is NP-complete, as FAS is fixed-parameter tractable in terms of the size of the feedback arc set [8]. Indeed, setting the size of the feedback arc-set to zero is equivalent to checking if the given graph is a DAG and the problem becomes solvable in polynomial time.

On the other hand, it has been shown that FAS is APX-hard, meaning that every problem in APX is reducible to it [25]. It also implies, assuming $\text{NP} \neq \text{P}$, that there is a constant $C \geq 1$ such that there is no polynomial time algorithm which provides a C -approximation. The reduction provided in this section implies:

► **Theorem 14.** *The WGV problem is APX-hard.*



■ **Figure 5** A bold edge in Figure 6 is actually $k + 1$ subdivided edges.

In addition, Guruswami et al. demonstrated that assuming the Unique Games Conjecture holds, and $\text{NP} \neq \text{P}$, there is no constant $C \geq 1$ such that a polynomial-time algorithm's approximate solution to FAS is always a factor C from the optimal solution. We state this as a lemma.

► **Lemma 15** (Guruswami et al. [20]). *Conditioned on the Unique Games Conjecture, for every $C \geq 1$, it is NP-hard to find a C -approximation to FAS.*

An approximation preserving reduction from FAS to WGV combined with Lemma 15 proves the other main result of this section:

► **Theorem 16.** *Conditioned on the Unique Games conjecture, for every constant $C \geq 1$, it is NP-hard to find a C -approximation to WGV.*

4.2 The Reduction of FAS to WGV

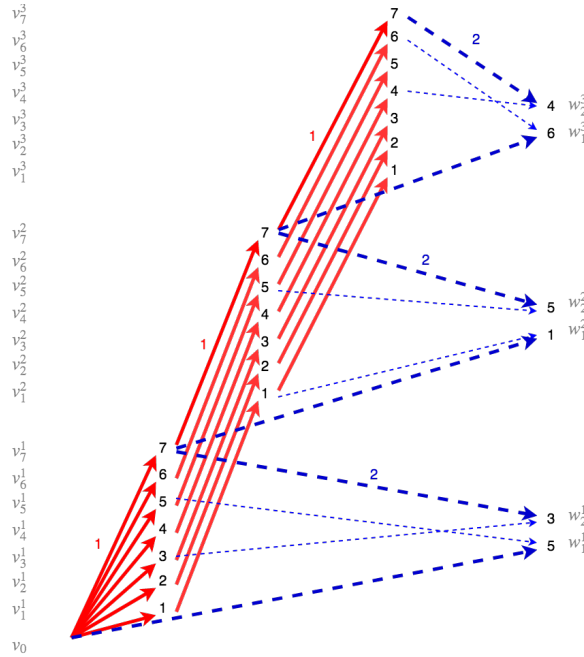
Let $T = t_1, t_2, \dots, t_n$ and inequalities $t_1^1 < t_2^1, t_1^2 < t_2^2, \dots, t_1^k < t_2^k$ be the input to FAS.

We define a heavy edge between the vertices u and v with label ℓ as $k + 1$ subdivided edges between u and v each with label ℓ . That is, a heavy edge between u and v with label ℓ consists of the edges (u, w_i, ℓ) and (w_i, v, ℓ) for $1 \leq i \leq k + 1$. See Figure 5 for an illustration. Use the following steps to create a graph (which is a DAG):

- Create a vertex v_0 and vertices v_i^j for $1 \leq i \leq n + 1$ and $1 \leq j \leq k$.
- For each inequality $t_1^j < t_2^j$ create a vertex for both t_1^j and t_2^j , labeled w_1^j and w_2^j , respectively.
- Create heavy edges $(v_0, v_i^1, 1)$ for $1 \leq i \leq n + 1$ and heavy edges $(v_i^j, v_i^{j+1}, 1)$ for $1 \leq i \leq n + 1, 1 \leq j \leq k - 1$.
- Create heavy edges $(v_0, w_1^1, 2)$, and heavy edges $(v_{n+1}^j, w_2^j, 2)$ and $(v_{n+1}^j, w_1^{j+1}, 2)$ for $1 \leq j \leq k - 1$, and heavy edge $(v_{n+1}^k, w_2^k, 2)$.
- Add the regular (not heavy) edges $(v_i^j, w_1^j, 2)$ if $t_i = t_1^j$, and $(v_i^j, w_2^j, 2)$ if $t_i = t_2^j$ for $1 \leq i \leq n, 1 \leq j \leq k$.

An example of the reduction is given in Figure 6. The intuition is that the vertices with an inbound heavy edge labeled 1 represent the permutation of the elements in T . The heavy edges labeled 1 force the permutation to be duplicated k times, once for each constraint. The vertices with the inbound edge label 2 represent the elements in each inequality. We will show that this is an approximation preserving reduction.

Let E' be a solution to WGV and $G' = (V, E \setminus E')$ and let π represent a proper ordering on the vertices of G' . Lemma 17 indicates that, other than permuting the ordering found on the vertices v_i^j for $1 \leq i \leq n$ (with the ordering duplicated for $1 \leq j \leq k$), the ordering for the vertices in Figure 6 is fixed.



■ **Figure 6** An example of the reduction from FAS to WGV where $T = 1, 2, 3, 4, 5, 6$ and the set of inequalities is $5 < 3$, $1 < 5$, and $6 < 4$.

► **Lemma 17.** *Let ϕ represent a permutation of the set $[n + 1]$. Any ordering π which provides a solution to the WGV instance is of the form*

$$v_0, v_{\phi(1)}^1, v_{\phi(2)}^1, \dots, v_{\phi(n+1)}^1, \dots, v_{\phi(1)}^k, v_{\phi(2)}^k, \dots, v_{\phi(n+1)}^k, w_1^1, w_2^1, w_1^2, w_2^2, \dots, w_1^k, w_2^k.$$

Proof. See full version [19]. ◀

Let $f(x)$ refer to the reduction described above applied to an instance x of FAS creating an instance $f(x)$ of WGV. We also refer to the solution to either of these problems as $\text{OPT}(\cdot)$, and $\text{val}(\cdot)$ as the cost function. For instance x of FAS $\text{val}(x)$ is the number of violated inequalities and for an instance $f(x)$ of WGV $\text{val}(f(x))$ it is the number of violating edges.

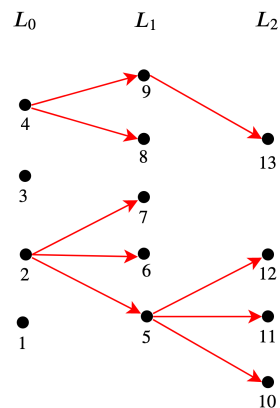
► **Lemma 18.** *Given an instance x of FAS, a solution (or sub-optimal solution) to the instance $f(x)$ of WGV that has $\ell \leq k$ axiom violating edges yields a solution (or sub-optimal solution) to x with ℓ violated inequalities. The converse holds as well.*

Proof. See full version [19]. ◀

► **Lemma 19.** *Given an instance x of FAS, a C -approximation to the solution $\text{OPT}(f(x))$ yields a C -approximation to the solution $\text{OPT}(x)$.*

Proof. By Lemma 18 any (sub-optimal) solution with objective value $C \cdot \text{val}(\text{OPT}(f(x)))$ to $f(x)$, gives us a (sub-optimal) solution to x with the same objective value, $C \cdot \text{val}(\text{OPT}(x))$. ◀

Theorem 14 follows from Lemma 19 and Theorem 16 follows from Lemma 19 and Lemma 15.



■ **Figure 7** Arborescences have their roots aligned in level L_0 . The relative ordering for each type of vertex can be read from top to bottom, left to right.

4.3 The Wheeler Subgraph Problem is in APX

The dual problem to WGV is the problem of finding the largest subgraph of G which is a Wheeler graph. This problem (defined in Section 1.2) is called Wheeler Subgraph problem, abbreviated WS. Unlike WGV, this problem yields a $\Theta(1)$ -approximate solution for constant σ .

We first prove the result for $\sigma = 1$. We then apply this result to get an approximation for $\sigma > 1$. The proof for $\sigma = 1$ uses a branching of a directed graph. A branching is a set of arborescence where an arborescence is a directed, rooted tree where all edges point away from the root. A branching is spanning in that every vertex in V is included exactly one arborescence in the branching.

► **Lemma 20.** *There exists a linear time $\Theta(1)$ -approximation algorithm for WS when the alphabet set size σ is one.*

Proof. Let V_0 be the set of sources in G (vertices with in-degree zero). There are two cases:

Case $|V_0| \leq n/2$: Take a branching \mathcal{F} of the input graph G such that each vertex with in-degree greater than zero is included in some non-singleton arborescence whose root is a source vertex in V_0 . Let $|\mathcal{F}|$ denote the total number of arborescences in \mathcal{F} . Since $|V_0| \leq n/2$, it follows that $|\mathcal{F}| \leq n/2$ as well.

We create a planar leveling (L_0, L_1, \dots) of \mathcal{F} by aligning all roots of the branching on level L_0 in an arbitrary order. Then set L_i to be all of the vertices which are distance i from some root in L_0 . Because these are trees, we can order the vertices in levels in such a way that the leveling is planar (and for the purpose of visualization say left to right as in Figure 7).

We claim that \mathcal{F} is a Wheeler graph and that we can obtain a proper ordering π for the vertices of \mathcal{F} from this leveling. Starting with V_0 , we order the vertices on each level from the bottom to top before proceeding right to the next level. One can check that the Wheeler graph axioms are satisfied.

The number of edges in \mathcal{F} , denoted $e(\mathcal{F})$, is equal to $n - |\mathcal{F}|$. And, since $|\mathcal{F}| \leq n/2$, we have that $e(\mathcal{F}) \geq n/2$. At the same time, by Theorem 5 the optimal number of edges, denoted $|E^*|$, is $\Theta(n)$. The ratio of the optimal solution value over the branching solution value is bounded. In particular, $|E^*|/e(\mathcal{F}) \leq \Theta(n)/(n/2) = \Theta(1)$. The construction of the branching, the planar leveling, and the extracting π can all be done in linear time.

51:14 On the Hardness and Inapproximability of Recognizing Wheeler Graphs

Case $|V_0| > n/2$: Take one outbound edge from each vertex in V_0 . We obtain a Wheeler graph with $|V_0| > n/2$ edges. This gives us a solution with an approximation ratio of $|E^*|/|V_0| < \Theta(n)/(n/2) = \Theta(1)$.

In either case, we have an approximate solution with \tilde{e} edges where $\tilde{e} \in \Theta(|E^*|)$. ◀

Next, we consider when $\sigma > 1$. Suppose $G^* = (V, E^*)$ is the optimal solution for G . Then $E^* = E_1^* \cup E_2^* \dots E_\sigma^*$ where $E_k^* = \{(u, v, k) \in E^*\}$. Let $G_k = (V, E_k)$ where $E_k = \{(u, v, k) \in E\}$ and let $G'_k = (V, E'_k)$ be the optimal solution for G_k . Then, since $|E_k^*| \leq |E'_k|$ we have

$$|E^*| = \sum_{k=1}^{\sigma} |E_k^*| \leq \sigma \cdot \max_k |E_k^*| \leq \sigma \cdot \max_k |E'_k|.$$

Applying the result for $\sigma = 1$ (Lemma 20), we can approximate $\max_k |E'_k|$ with a solution having $\tilde{e} = \alpha \cdot \max_k |E'_k|$ edges for some constant $\alpha \leq 1$. Therefore,

$$\frac{\alpha}{\sigma} |E^*| \leq \alpha \max_k |E'_k| = \tilde{e} \leq \max_k |E'_k| \leq |E^*|.$$

So the solution provides $\Omega(1/\sigma)$ -approximation for G as well.

► **Theorem 21.** *There exists a linear time $\Omega(1/\sigma)$ -approximation algorithm for WS.*

As a final result, we note that the algorithm presented in Section 3 also provides us with an exponential time solution to the two optimization problems defined in Section 4. The solution is to iterate over all possible subsets of edges in E , take the corresponding induced subgraph, and apply Algorithm 1 to identify if the induced subgraph is isomorphic to a Wheeler graph. For both the WGV and WS problems the optimal solution is the encoding with the fewest edges removed. The resulting time complexity is the same as in Theorem 10 with the addition of one e term in the exponent. We have shown the following:

► **Theorem 22.** *The WGV problem and WS problem for an input $G = (V, E)$ with $n = |V|$, $e = |E|$ and σ is the size of the edge label alphabet can be solved in time $2^{e \log \sigma + O(n+e)}$.*

References

- 1 Alfred V. Aho and Margaret J. Corasick. Efficient String Matching: An Aid to Bibliographic Search. *Commun. ACM*, 18(6):333–340, 1975. doi:10.1145/360825.360855.
- 2 Jarno Alanko, Travis Gagie, Gonzalo Navarro, and Louisa Seelbach Benkner. Tunneling on Wheeler Graphs. *CoRR*, abs/1811.02457, 2018. arXiv:1811.02457.
- 3 Jarno Alanko, Alberto Policriti, and Nicola Prezza. On Prefix-Sorting Finite Automata, 2019. arXiv:1902.01088.
- 4 László Babai and Eugene M. Luks. Canonical Labeling of Graphs. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 171–183, 1983. doi:10.1145/800061.808746.
- 5 Djamel Belazzougui. Succinct Dictionary Matching with No Slowdown. In *Combinatorial Pattern Matching, 21st Annual Symposium, CPM 2010, New York, NY, USA, June 21-23, 2010. Proceedings*, pages 88–100, 2010. doi:10.1007/978-3-642-13509-5_9.
- 6 Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn Graphs. In *Algorithms in Bioinformatics - 12th International Workshop, WABI 2012, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 225–235, 2012. doi:10.1007/978-3-642-33122-0_18.
- 7 Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm, 1994.

- 8 Jianer Chen, Yang Liu, Songjian Lu, Barry O’Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):21:1–21:19, 2008. doi:10.1145/1411509.1411511.
- 9 Francisco Claude, Gonzalo Navarro, and Alberto Ordóñez Pereira. The wavelet matrix: An efficient wavelet tree for large alphabets. *Inf. Syst.*, 47:15–32, 2015. doi:10.1016/j.is.2014.06.002.
- 10 Nicolaas Govert De Bruijn. A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen*, 49(49):758–764, 1946.
- 11 Vida Dujmovic and David R. Wood. On Linear Layouts of Graphs. *Discrete Mathematics & Theoretical Computer Science*, 6(2):339–358, 2004. URL: <http://dmtcs.episciences.org/317>.
- 12 Massimo Equi, Roberto Grossi, and Veli Mäkinen. On the Complexity of Exact Pattern Matching in Graphs: Binary Strings and Bounded Degree. *CoRR*, abs/1901.05264, 2019. arXiv:1901.05264.
- 13 Massimo Equi, Roberto Grossi, Alexandru I. Tomescu, and Veli Mäkinen. On the Complexity of Exact Pattern Matching in Graphs: Determinism and Zig-Zag Matching. *CoRR*, abs/1902.03560, 2019. arXiv:1902.03560.
- 14 Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57(1):4:1–4:33, 2009. doi:10.1145/1613676.1613680.
- 15 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.
- 16 Paolo Ferragina and Rossano Venturini. The compressed permuterm index. *ACM Trans. Algorithms*, 7(1):10:1–10:21, 2010. doi:10.1145/1868237.1868248.
- 17 Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for BWT-based data structures. *Theor. Comput. Sci.*, 698:67–78, 2017. doi:10.1016/j.tcs.2017.06.016.
- 18 Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. pBWT: Achieving Succinct Data Structures for Parameterized Pattern Matching and Related Problems. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 397–407, 2017. doi:10.1137/1.9781611974782.25.
- 19 Daniel Gibney and Sharma V. Thankachan. On the Hardness and Inapproximability of Recognizing Wheeler Graphs, 2019. arXiv:1902.01960.
- 20 Venkatesan Guruswami, Rajsekar Manokaran, and Prasad Raghavendra. Beating the Random Ordering is Hard: Inapproximability of Maximum Acyclic Subgraph. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 573–582, 2008. doi:10.1109/FOCS.2008.51.
- 21 Lenwood S. Heath and Sriram V. Pemmaraju. Stack and Queue Layouts of Directed Acyclic Graphs: Part II. *SIAM J. Comput.*, 28(5):1588–1626, 1999. doi:10.1137/S0097539795291550.
- 22 Lenwood S. Heath, Sriram V. Pemmaraju, and Ann N. Trenk. Stack and Queue Layouts of Directed Acyclic Graphs: Part I. *SIAM J. Comput.*, 28(4):1510–1539, 1999. doi:10.1137/S0097539795280287.
- 23 Lenwood S. Heath and Arnold L. Rosenberg. Laying out Graphs Using Queues. *SIAM J. Comput.*, 21(5):927–958, 1992. doi:10.1137/0221055.
- 24 Wing-Kai Hon, Tsung-Han Ku, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Faster compressed dictionary matching. *Theor. Comput. Sci.*, 475:113–119, 2013. doi:10.1016/j.tcs.2012.10.050.
- 25 Viggo Kann. *On the approximability of NP-complete optimization problems*. PhD thesis, Royal Institute of Technology Stockholm, 1992.
- 26 Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. An Extension of the Burrows Wheeler Transform and Applications to Sequence Comparison and Data Compression. In *Combinatorial Pattern Matching, 16th Annual Symposium, CPM 2005, Jeju Island, Korea, June 19-22, 2005, Proceedings*, pages 178–189, 2005. doi:10.1007/11496656_16.

51:16 On the Hardness and Inapproximability of Recognizing Wheeler Graphs

- 27 Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. An extension of the Burrows-Wheeler Transform. *Theor. Comput. Sci.*, 387(3):298–312, 2007. doi:10.1016/j.tcs.2007.07.014.
- 28 Gary L. Miller. Graph Isomorphism, General Remarks. *J. Comput. Syst. Sci.*, 18(2):128–142, 1979. doi:10.1016/0022-0000(79)90043-6.
- 29 Adam M. Novak, Erik Garrison, and Benedict Paten. A graph extension of the positional Burrows-Wheeler transform and its applications. *Algorithms for Molecular Biology*, 12(1):18:1–18:12, 2017. doi:10.1186/s13015-017-0109-9.
- 30 Jaroslav Opatrny. Total Ordering Problem. *SIAM J. Comput.*, 8(1):111–114, 1979. doi:10.1137/0208008.
- 31 Jouni Sirén, Niko Välimäki, and Veli Mäkinen. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 11(2):375–388, 2014.
- 32 D Younger. Minimum feedback arc sets for a directed graph. *IEEE Transactions on Circuit Theory*, 10(2):238–245, 1963.

Evaluation of a Flow-Based Hypergraph Bipartitioning Algorithm

Lars Gottesbüren

Institute of Theoretical Informatics, Karlsruhe Institute of Technology, Germany
lars.gottesbueren@kit.edu

Michael Hamann

Institute of Theoretical Informatics, Karlsruhe Institute of Technology, Germany
michael.hamann@kit.edu

Dorothea Wagner

Institute of Theoretical Informatics, Karlsruhe Institute of Technology, Germany
dorothea.wagner@kit.edu

Abstract

In this paper, we propose HyperFlowCutter, an algorithm for balanced hypergraph bipartitioning that is based on minimum S - T hyperedge cuts and maximum flows. It computes a sequence of bipartitions that optimize cut size and balance in the Pareto sense, being able to trade one for the other. HyperFlowCutter builds on the FlowCutter algorithm for partitioning graphs. We propose additional features, such as handling disconnected hypergraphs, novel methods for obtaining starting S, T pairs as well as an approach to refine a given partition with HyperFlowCutter. Our main contribution is ReBaHFC, a new algorithm which obtains an initial partition with the fast multilevel hypergraph partitioner PaToH and then improves it using HyperFlowCutter as a refinement algorithm. ReBaHFC is able to significantly improve the solution quality of PaToH at little additional running time. The solution quality is only marginally worse than that of the best-performing hypergraph partitioners KaHyPar and hMETIS, while being one order of magnitude faster. Thus ReBaHFC offers a new time-quality trade-off in the current spectrum of hypergraph partitioners. For the special case of perfectly balanced bipartitioning, only the much slower plain HyperFlowCutter yields slightly better solutions than ReBaHFC, while only PaToH is faster than ReBaHFC.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Hypergraph Partitioning, Maximum Flows, Algorithm Engineering

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.52

Related Version A full version of the paper is available at <https://arxiv.org/abs/1907.02053>.

Supplement Material Source code at <https://github.com/kit-algo/HyperFlowCutter>

Funding This work was supported by the DFG under grants WA654/19-2 and WA654/22-2. The authors acknowledge support by the state of Baden-Württemberg through bwHPC.

Acknowledgements We thank Ben Strasser, Tim Zeitz and Lukas Barth for helpful discussions.

1 Introduction

Given a hypergraph $H = (V, E)$, a hyperedge cut $C \subset E$ is a set of hyperedges whose removal disconnects H . The *balanced hypergraph bipartitioning problem* is to find a hyperedge cut of minimum cardinality whose removal separates H into two blocks of roughly equal size – up to $(1 + \varepsilon)\frac{|V|}{2}$. Hypergraph partitioning has applications in VLSI design, database sharding, and high performance computing, in particular load balancing and reducing communication for highly parallel computations as well as accelerating sparse matrix vector multiplications. This problem is NP-hard [35] and it is hard to approximate, even for graphs [12]. Therefore, practical algorithms use heuristics. Most of them are based on the *multilevel* framework [28].



© Lars Gottesbüren, Michael Hamann, and Dorothea Wagner;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 52; pp. 52:1–52:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper we consider a different approach based on the max-flow min-cut duality. The basic idea has already been used in the Flow-Balanced-Bipartition algorithm (FBB) of Yang and Wong [51]. So far it has not been of further consideration due to being too slow compared to multilevel algorithms and too slow to solve current instances in feasible time. More recently, FlowCutter [27] for graph bipartitioning has been introduced independently of FBB. It is designed for computing very small node separators in road networks with a rather loose balance constraint; $\varepsilon = 0.33$ is recommended for the application of accelerating shortest path computations [20]. Based on similar ideas as FBB but equipped with more engineering, it computes both unbalanced and highly balanced bipartitions of high quality on the Walshaw graph partitioning benchmark [47].

Contribution. We present HyperFlowCutter, an algorithm which computes a series of hypergraph bipartitions with increasing balance, up to perfectly balanced solutions. With HyperFlowCutter, we extend FlowCutter to hypergraphs and contribute additional features. We provide a method to handle disconnected hypergraphs, which FlowCutter and FBB cannot handle. Our main contribution is ReBaHFC, an algorithm to refine a given partition using HyperFlowCutter. It is a natural extension of the max-flow based refinement of the k -way hypergraph partitioner KaHyPar [29]. We provide a thoroughly engineered implementation as well as an extensive experimental evaluation on the benchmark set of Heuer and Schlag [30], comparing HyperFlowCutter and ReBaHFC against the state-of-the-art hypergraph partitioning tools KaHyPar [30, 29], PaToH [13] and hMETIS [31, 32].

In our experiments we use the fast algorithm PaToH to obtain initial partitions for ReBaHFC. When using the quality preset of PaToH, ReBaHFC computes solutions for $\varepsilon = 0.03$, which are only slightly worse than those of the best-performing partitioners KaHyPar and hMETIS and significantly better than those of PaToH. ReBaHFC is only marginally slower than PaToH and thus, like PaToH, it is one order of magnitude faster than KaHyPar and hMETIS, when using the quality preset, and two orders of magnitude faster, when using the default preset. Furthermore, ReBaHFC with the PaToH default preset computes better solutions than PaToH with its quality preset. Thus ReBaHFC offers new time-quality trade-offs. For the special case of perfectly balanced bipartitioning, only the much slower plain HyperFlowCutter yields marginally better solutions than ReBaHFC, while only PaToH is faster than ReBaHFC.

Outline. After discussing related work in Section 2 and presenting notation and preliminaries in Section 3, we introduce the core algorithm of HyperFlowCutter for S - T hyperedge cuts in Section 4.1. Then we show how to handle disconnected hypergraphs in Section 4.2, propose our refinement algorithm ReBaHFC in Section 4.3 and finally discuss the experimental evaluation in Section 5.

2 Related Work

For an overview of the field of hypergraph partitioning we refer to survey articles [7, 40, 5]. The most common approach among hypergraph partitioning tools is the multilevel framework. Multilevel algorithms repeatedly *contract* vertices to obtain a hierarchy of *coarser* hypergraphs while trying to preserve the cut structure. On the coarsest hypergraph an *initial partition* is computed in some way. Then the contractions are reversed step-by-step and after every uncontraction a *refinement* algorithm tries to improve the solution. Most multilevel algorithms use a variant of the Fiduccia-Mattheyses (FM) [23] or Kernighan-Lin (KL) [33] local vertex

moving heuristics. These algorithms move vertices between blocks, prioritized by cut improvement. The multilevel framework has been immensely successful because it provides a global view on the problem through local operations on the coarse levels. Furthermore, it allows a great deal of engineering and tuning, which have a drastic impact on solution quality. Even though this framework has been used since the 1990s, the implementations are still improving today. A selection of well-known multilevel hypergraph partitioners are PaToH [13] (scientific computing), hMETIS [31, 32] (VLSI design) KaHyPar [30, 29] (general purpose, n-level), Zoltan [19] (scientific computing, parallel), Zoltan-AlgD [46] (algebraic distances based coarsening, sequential), Mondriaan [49] (sparse matrices), MLPart [4] (circuit partitioning) and Parkway [48] (parallel).

Compared to graph partitioning, the performance of local vertex moving suffers from the presence of large hyperedges with vertices scattered over multiple blocks, since many moves have zero cut improvement. On coarse levels of the multilevel hierarchy, this problem is alleviated since hyperedges contain fewer vertices. A second remedy are flow-based refinement algorithms. For graphs, Sanders and Schulz [43] extract a size-constrained corridor around the cut and compute a minimum cut within this corridor. If the cut is balanced, an improved solution was found, otherwise the step is repeated with a smaller corridor. Heuer et al. [29] extend their approach to hypergraphs by using *Lawler networks* [34]. The Lawler network of a hypergraph is a flow network such that minimum S - T hyperedge cuts can be computed via max-flow.

In their Flow-Balanced-Bipartition algorithm (FBB), Yang and Wong [51] use incremental maximum flows on the Lawler network to compute ε -balanced hypergraph bipartitions. Liu and Wong [37] enhance FBB with a *most-desirable-minimum-cut* heuristic, which is inspired by the correspondence between S - T minimum cuts and closed node sets due to Picard and Queyranne [41]. It is similar to the *most-balanced-minimum-cut* heuristics used in the multilevel graph partitioning tool KaHiP [43] and KaHyPar-MF [29] as well as the *avoid-augmenting-paths* piercing heuristics of FlowCutter [27] and HyperFlowCutter. Li et al. [36] propose a push-relabel algorithm, which operates directly on the hypergraph. Furthermore they present heuristics rooted in VLSI design for choosing sets of initial seed vertices S and T as well as piercing vertices. The performance of their approach in other contexts than VLSI design remains unclear.

For perfectly balanced graph partitioning, diffusion-based methods have been successful [38]. Furthermore Sanders and Schulz [44] propose an algorithm based on detecting negative cycles, which is used on top of their evolutionary partitioner. Delling and Werneck [18] provide an efficient implementation of an optimal branch-and-bound algorithm. Additionally there are metaheuristic approaches such as PROBE [14], as well as multilevel memetic algorithms due to Benlic and Hao [9, 10, 11].

3 Preliminaries

A *hypergraph* $H = (V, E)$ consists of a set of n vertices V and a set of m hyperedges E , where a hyperedge e is a subset of the vertices V . A vertex $v \in V$ is *incident* to hyperedge $e \in E$ if $v \in e$. The vertices incident to e are called the *pins* of e . We denote the incident hyperedges of v by $I(v)$ and its degree by $\deg(v) := |I(v)|$. Furthermore let $p := \sum_{e \in E} |e|$ denote the total number of pins in H . All hypergraphs in this paper are unweighted. H can be represented as a bipartite graph $G = (V \cup E, \{(v, e) \in V \times E \mid v \in e\})$ with bipartite node set $V \cup E$ and an edge for every pin. This is also referred to as the *star expansion* of H . H is *connected* if its star expansion is connected. Let $V[E'] := \bigcup_{e' \in E'} e'$ denote the vertex set induced by the hyperedge set E' . To avoid confusion, we use the terms vertices, hyperedges and pins for hypergraphs, and we use the terms nodes and edges for graphs.

3.1 Hypergraph Partitioning

A *bipartition* of a hypergraph H is a partition (A, B) of the vertices V into two non-empty, disjoint sets (called blocks). The *cut* $\text{cut}(A, B) := \{e \in E \mid e \cap A \neq \emptyset \wedge e \cap B \neq \emptyset\}$ consists of all hyperedges with pins in both blocks. The *size* of a cut is the number of cut hyperedges $|\text{cut}(A, B)|$. Let $\varepsilon \in [0, 1)$. A bipartition (A, B) is ε -balanced if $\max(|A|, |B|) \leq \lceil (1 + \varepsilon) \frac{n}{2} \rceil$. The *balanced hypergraph bipartitioning problem* is to find an ε -balanced bipartition (A, B) which minimizes the cut. The special case $\varepsilon = 0$ is called *perfectly balanced bipartitioning*.

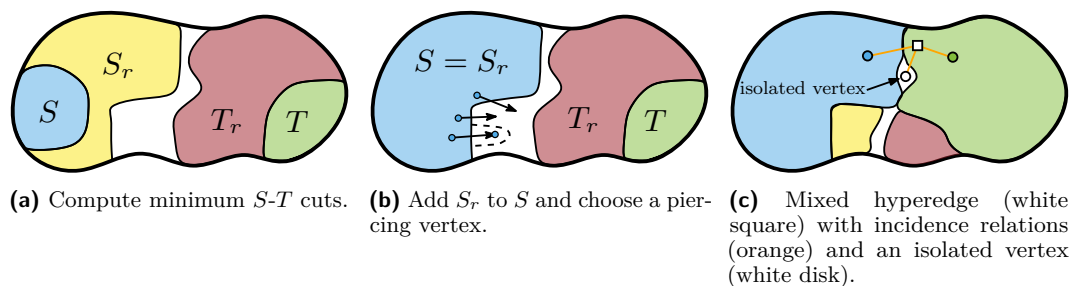
3.2 Maximum Flows

A flow network $\mathcal{N} = (\mathcal{V}, \mathcal{E}, S, T, c)$ is a simple symmetric directed graph $(\mathcal{V}, \mathcal{E})$ with two non-empty *terminal* node sets $S, T \subsetneq \mathcal{V}$, $S \cap T = \emptyset$, the source and target node set, as well as a capacity function $c : \mathcal{E} \mapsto \mathbb{R}_{\geq 0}$. Any node that is not a source node and not a target node is a *non-terminal* node. A flow in \mathcal{N} is a function $f : \mathcal{E} \mapsto \mathbb{R}$ subject to the *capacity constraint* $f(e) \leq c(e)$ for all edges e , *flow conservation* $\sum_{(u,v) \in \mathcal{E}} f((u,v)) = 0$ for all non-terminal nodes v and *skew symmetry* $f((u,v)) = -f((v,u))$ for all edges (u,v) . The *value* of a flow $|f| := \sum_{s \in S, (s,u) \in \mathcal{E}} f((s,u))$ is the amount of flow leaving S . The *residual capacity* $r_f(e) := c(e) - f(e)$ is the additional amount of flow that can pass through e without violating the capacity constraint. The residual network with respect to f is the directed graph $\mathcal{N}_f = (\mathcal{V}, \mathcal{E}_f)$ where $\mathcal{E}_f := \{e \in \mathcal{E} \mid r_f(e) > 0\}$. A node v is *source-reachable* if there is a path from S to v in \mathcal{N}_f , it is *target-reachable* if there is a path from v to T in \mathcal{N}_f . We denote the source-reachable and target-reachable nodes by S_r and T_r , respectively. An *augmenting path* is an S - T path in \mathcal{N}_f . The flow f is a *maximum flow* if $|f|$ is maximal of all possible flows in \mathcal{N} . This is the case iff there is no augmenting path in \mathcal{N}_f . An S - T edge cut is a set of edges whose removal disconnects S and T . The value of a maximum flow equals the weight of a minimum-weight S - T edge cut [24]. The *source-side cut* consists of the edges from S_r to $\mathcal{V} \setminus S_r$ and the *target-side cut* consists of the edges from T_r to $\mathcal{V} \setminus T_r$. The bipartition $(S_r, \mathcal{V} \setminus S_r)$ is induced by the source-side cut and $(\mathcal{V} \setminus T_r, T_r)$ is induced by the target-side cut. Note that $\mathcal{V} \setminus S_r \setminus T_r$ is not necessarily empty.

3.3 Hyperedge Cuts via Maximum Flows

Lawler [34] uses maximum flows to compute minimum S - T hyperedge cuts without balance constraints. On the star expansion, the standard construction to model node capacities as edge capacities [1] is applied to the hyperedge-nodes. A hyperedge e is expanded into an *in-node* e_i and an *out-node* e_o joined by a directed *bridge edge* (e_i, e_o) with unit capacity. For every pin $u \in e$ there are two directed *external edges* $(u, e_i), (e_o, u)$ with infinite capacity. The transformed graph is called the *Lawler network*. A minimum S - T edge cut in the Lawler network consists only of bridge edges, which directly correspond to S - T cut hyperedges in H .

Instead of using the Lawler network, we emulate max-flow algorithms directly on the hypergraph, using an approach first proposed by Pistorius and Minoux [42]. In the paper it is formulated for unit weight hyperedges and the Edmonds-Karp flow algorithm [22] but it can be easily extended to handle weighted hyperedges and emulate any flow algorithm. For every hyperedge $e \in E$, we store the pins sending and receiving flow via e . In this work, we consider only unit weight hyperedges and thus need to store only one pin $\text{flow-from}(e)$ sending flow into e , and one pin $\text{flow-to}(e)$ receiving flow from e . To keep the description simple, it relies on this assumption as well. Let u be a fixed vertex. The idea is to enumerate short paths of the form $u \rightarrow e \in I(u) \rightarrow v \in e$ that correspond to paths in the residual Lawler network. This allows us to emulate algorithms for traversing the residual Lawler network directly on



■ **Figure 1** Flow augmentation and computing S_r, T_r in Fig.1a; adding S_r to S and piercing the source-side cut in Fig.1b. S in blue, $S_r \setminus S$ in yellow, T in green, $T_r \setminus T$ in red, $V \setminus S_r, \setminus T_r$ in white.

the hypergraph, such as Breadth-First-Search or Depth-First-Search, as well as other types of local operations, e. g., the *discharge* and *push* operations in push-relabel algorithms [25]. For every hyperedge $e \in I(u)$ we do the following. If e has no flow, we enumerate all pins $v \in e$. These paths correspond to (u, e_i, e_o, v) in the Lawler network. If e has flow and $u = \text{flow-to}(e)$ we also enumerate all pins $v \in e$. However, these paths correspond to (u, e_o, e_i, v) in the Lawler network. If e has flow and $u = \text{flow-from}(e)$, there is no path in the residual Lawler network starting at u that uses e . If e has flow and $\text{flow-from}(e) \neq u \neq \text{flow-to}(e)$, we enumerate just one path $(u, e, \text{flow-from}(e))$, corresponding to $(u, e_i, \text{flow-from}(e))$ in the Lawler network. If we can push flow from the vertex $\text{flow-from}(e)$ to T , we can redirect the flow that the vertex $\text{flow-from}(e)$ sends into e , and instead route flow coming from u to $\text{flow-to}(e)$. Then u becomes the new $\text{flow-from}(e)$.

We use this approach in our implementation because it is significantly more efficient than the Lawler network in practice. In the last case we can avoid scanning all pins of e . In a preliminary experiment, computing flow directly on the hypergraph yielded a speedup of 15 over using the Lawler network, for a hypergraph with maximum hyperedge size of only 36. The speedup will be more extreme on hypergraphs with larger hyperedges.

Via the Lawler network and the above emulation approach, the notions of flow, source-reachable vertices and source-side cuts translate naturally from graphs to hypergraphs. We use the notation and terminology already known from max-flows in graphs.

4 HyperFlowCutter

In the following we outline the core HyperFlowCutter algorithm, which can only be used on connected hypergraphs. Then we discuss how to handle disconnected hypergraphs, and finally show how to improve an existing partition using HyperFlowCutter.

4.1 The Core Algorithm

The idea of the *core* HyperFlowCutter algorithm is to solve a sequence of incremental S - T max-flow min-cut problems with monotonically increasing cut size and balance, until an ε -balanced bipartition is found. They are incremental in the sense that the terminals S, T of the current flow problem are subsets of the terminals in the next flow problem, which allows us to reuse previously computed flows.

Given starting terminal sets $S_{\text{init}}, T_{\text{init}}$, we set $S := S_{\text{init}}, T := T_{\text{init}}$. First, we compute a maximum S - T flow. We terminate if the bipartition $(S_r, V \setminus S_r)$ induced by the source-side cut or $(V \setminus T_r, T_r)$ induced by the target-side cut is ε -balanced. Otherwise, we add the source-reachable vertices S_r to S , if $|S_r| \leq |T_r|$, or we add T_r to T if $|S_r| > |T_r|$. Assume

$|S_r| \leq |T_r|$ without loss of generality. Further, we add one or more vertices, called *piercing vertices*, to S . This step is called *piercing* the cut. It ensures that the next flow problem yields a different bipartition. Subsequently, we augment the previous flow to a max-flow that respects the new sources. We repeat these steps until an ε -balanced bipartition is found.

Note that after adding S_r to S , the flow is still a maximum S - T flow, even though the added vertices are now exempt from flow conservation. Using the smaller side allows it to catch up with the larger side. In particular, this ensures that ε -balance is always possible, as neither side grows beyond $\lceil n/2 \rceil$ vertices.

A hyperedge with pins in both S and T is *mixed*, all other hyperedges are *non-mixed*. We consider two options to find piercing vertices. The preferred option is to choose all pins $e \setminus S \setminus T$ of a non-mixed hyperedge e in the source-side cut. Adding multiple vertices is faster in practice. This small detail is a major difference to FBB [51] and is necessary to make the running time feasible on the used benchmark set. If the source-side cut contains only mixed hyperedges, we choose a single non-terminal vertex incident to the source-side cut. We prefer piercing vertices which are not reachable from T , as these avoid augmenting paths in the next iteration, and thus the cut size does not increase. Avoiding augmenting paths has the highest precedence, followed by choosing hyperedges over single vertices. Ties are broken randomly. If the piercing vertices are not reachable from T , we do not recompute T_r and we skip flow augmentation, but we do recompute S_r .

We experimented with other piercing methods, including trying to avoid mixed hyperedges, the distance-based piercing of FlowCutter, as well as piercing based on a machine learning technique named ensemble classification. We discuss ensemble classification again in the experimental section, although in a different context. None of these approaches yielded consistent or significant quality improvements over just avoiding augmenting paths and random tie-breaking, which is why we use only those.

Asymptotic Complexity. The asymptotic running time of Core HyperFlowCutter is $\mathcal{O}(cp)$ where c is the cut size of the ε -balanced partition and p is the number of pins in the hypergraph. Roughly speaking, the term p stems from performing up to one hypergraph traversal per flow unit. Here we use that $n \leq p, m \leq p$ holds for connected hypergraphs. Note that the running time is output-sensitive, however the factor c is rather pessimistic in practice, since the flow algorithm finds many augmenting paths in a single traversal. The original proof for FlowCutter [27] is applicable, but implementing the piercing step requires a little care. Selecting piercing vertices takes $\mathcal{O}(c)$ per iteration, and we have at most $n \leq p$ iterations. Selecting a non-mixed hyperedge for piercing takes $\mathcal{O}(c)$ time, by iterating over the cut hyperedges. Selecting single piercing vertices which avoid augmenting paths whenever possible, is slightly more involved, when restricted to $\mathcal{O}(c)$ time. For every cut hyperedge e we additionally track its pins that are not reachable from either side. This can be implemented with a linked list, from which we delete vertices once they get reachable from a side. An alternative implementation divides the memory storing the pins of e into three regions: the pins in S_r , in T_r or not reachable. Then we can check, whether e has pins that are not reachable from either side, and if so pick one. In practice, this adds significantly to the complexity of the implementation and the piercing step is never critical for running time, so our implementation simply scans all non-terminal boundary vertices.

Our implementation has $\mathcal{O}(n + m)$ memory footprint by transferring and re-engineering the implementation details from [27]. This is dominated by the $\mathcal{O}(n + m + p)$ memory for storing the hypergraph.

Isolated Vertices. We call a vertex $v \notin S \cup T$ *isolated* if every incident hyperedge $e \in I(v)$ is mixed. Figure 1c illustrates an isolated vertex. An isolated vertex cannot be reached from either S or T via hyperedges not in the cut. Mixed hyperedges remain in both the source-side cut and the target-side cut. Thus isolated vertices can be moved freely between the two blocks to increase balance. It never makes sense to permanently add them to a side, so we exclude them from the piercing step. Furthermore, this needs to be reflected when checking for ε -balance. For checking the bipartition $(S_r, V \setminus S_r)$ we assume up to $n/2 - |S_r|$ of the isolated vertices were part of S_r , analogously for T_r .

Maximum Flow Algorithm. In our implementation, we adapt Dinic maximum flow algorithm [21] to operate directly on the hypergraph, as described in Section 3.3. Dinic algorithm has two alternating phases that are repeated until a maximum flow is found: computing hop distance labels of nodes, using Breadth-First-Search, and finding augmenting paths using Depth-First-Search, such that the distance labels always increase by one along the path. We interleave the first phase of Dinic algorithm with the computation of reachable vertices, in order to avoid duplicate hypergraph traversal. This intrusive optimization is important for improving the running time of flow augmentation in practice, as the part of the running time of the flow algorithm that cannot be charged towards computing reachable vertices is dominated by the part that can be. This is not possible with push-relabel algorithms [25], which is why they were ruled out after short experimentation. We experimented with the Edmonds-Karp flow algorithm [22], modified to augment flow along multiple vertex-disjoint paths in one Breadth-First-Search by propagating vertex labels through the layers. It is slightly slower than Dinic for plain HyperFlowCutter but unfeasible for the refinement variant of HyperFlowCutter, since there are fewer vertices to route flow through and thus the amount of flow augmented per Breadth-First-Search is limited by a few bottleneck vertices.

4.2 Disconnected Hypergraphs

The HyperFlowCutter core algorithm is limited to connected hypergraphs since it computes S - T -cuts. An obvious approach for handling disconnected hypergraphs is connecting components artificially. We refrained from this because a component that intersects neither S nor T would be added to S or T in its entirety. Instead, we run the core algorithm up to $\varepsilon = 0$ on every connected component. The core algorithm computes multiple bipartitions with different balances. We systematically try all possible ways to combine the bipartitions of the components into a bipartition of H .

This can be stated as a generalization of the well-known SUBSETSUM problem. SUBSETSUM is a weakly NP-hard decision problem, which asks whether a subset of an input multiset of positive integers $A = \{a_1, \dots, a_z\}$, the *items*, sums to an input target sum W . Finding a bipartition with zero cut is equivalent to SUBSETSUM, where the items are the sizes of the components and W is the minimum size of the smaller block. We are interested in any subset summing to at least W . Let A be sorted in increasing order and let $Q(i, S)$ be a boolean variable, which is true iff a subset of the first i items sums to S . The standard pseudo-polynomial time dynamic program (DP) [15, Section 35.5] for SUBSETSUM computes solutions for all possible target sums. It fills the DP table Q by iterating through the items in increasing order and setting $Q(i, S)$ to true if $Q(i-1, S - a_i)$ or $Q(i-1, S)$ is true. For filling row i , only row $i-1$ is required, so the memory footprint is not quadratic.

We now turn to non-zero cut bipartitions by allowing to split items in different ways and associating costs with the splits. The core algorithm computes multiple bipartitions P_i on component C_i , at most one for every possible size of the smaller side. These correspond

directly to the different ways we can split items. The associated cost is the cut size. We modify the standard SUBSETSUM DP to minimize the added cuts instead of finding any subset, ensuring every component/item is split only one way in a solution, i. e., select a bipartition, and trying the smaller and larger side of the component bipartition for the smaller side of the bipartition of H . The worst case asymptotic running time of this DP is $\mathcal{O}(\sum_{i=1}^z \sum_{j=1}^{i-1} |P_i||P_j|)$.

We propose some optimizations to make the approach faster in practice. First we solve standard SUBSETSUM to check whether H has an ε -balanced bipartition with zero cut. For the *gap-filler* optimization, we find the largest $g \in \mathbb{N}$ such that for every $x \in [0, g]$ there are connected components, whose sizes sum to x . Computing g is possible in $\mathcal{O}(n)$ time. Let C_1, \dots, C_z be sorted by increasing size, which takes $\mathcal{O}(n)$ time using counting sort [15, Section 8.2]. Then $g = \sum_{j=1}^{k-1} |C_j|$ for the smallest k such that $|C_k| > 1 + \sum_{j=1}^{k-1} |C_j|$. It is never beneficial to split the components C_1, \dots, C_{k-1} . For most hypergraphs we consider in our experiments, we do not invoke the DP because, due to gap-filler, we split only the largest component. For the hypergraphs on which we do invoke the DP, its running time is negligible. Nonetheless, it is easy to construct a worst case instance, where the quadratic running time is prohibitive. For a robust algorithm, we propose to sample bipartitions from every P_i so that the worst case running time falls below some input threshold. The samples should include balanced bipartitions to guarantee that a balanced partition on H can be combined from those on $H[C_i]$.

4.3 HyperFlowCutter as a Refinement and Balancing Algorithm

Instead of partitioning from scratch, HyperFlowCutter can be used to refine an existing balanced bipartition $\pi = (V_1, V_2)$, or repair the balance of an unbalanced bipartition. We fix two blocks of vertices $V'_i \subset V_i$ such that $|V'_i| \leq \alpha \cdot n$ for a *relative block-size threshold* parameter $\alpha \in [0, 0.5]$. To obtain V'_i we run Breadth-First-Search from the boundary vertices on the side of V_i until $|V_i| - \alpha \cdot n$ vertices have been visited. The $\alpha \cdot n$ vertices not visited by the Breadth-First-Search are set as V'_i . Then we run HyperFlowCutter with $S = V'_1, T = V'_2$. We call this algorithm *ReBaHFC*. This idea is equivalent to the way KaHyPar and KaHiP extract *corridors* around the cut for their flow-based refinement. Only the semantics of the size constraint are different to our approach. However, KaHyPar and KaHiP only compute one flow. If the associated bipartition is not balanced, a smaller flow network is derived. This is repeated until the bipartition is balanced. ReBaHFC does not need to repeatedly re-scale flow networks.

In this work we only perform refinement as a post-processing step to a given partition, whereas KaHyPar and KaHiP employ flow-based refinement on the multilevel hierarchy. In a future work we hope to integrate HyperFlowCutter based refinement into KaHyPar. Using ReBaHFC could eliminate the significant overhead of repeated rescaling and improve solution quality – in particular when the minimum cut is just short of being balanced.

We use the fast multilevel partitioner PaToH [13] to obtain initial partitions. We briefly discuss properties of PaToH and differences between its presets. For coarsening, PaToH uses agglomerative clustering, based on the number of common hyperedges divided by cluster size. For initial partitioning, a portfolio of graph growing, bin packing and neighborhood expansion algorithms is used. For refinement PaToH uses a pass of FM [23] followed by a pass of KL [33], each initialized with boundary nodes.

In order to improve cut size, Walshaw [50] proposed to iterate the multilevel scheme by contracting only nodes from the same block, which maintains the cut size, thus allowing refinement on coarse levels starting from a relatively high quality partition. The existing

■ **Table 1** Average and quantile speedups of the hybrid and interleaved execution strategies.

	avg	min	0.1	0.25	median	0.75	0.9	max
hybrid	2.21	0.4	0.96	1.07	1.29	1.55	2.57	49.66
interleaved	3.88	0.55	1.0	1.14	1.38	1.74	2.66	175.47

partition serves as initial partition on the coarsest level. One iteration is called a *V-cycle*. Contraction can be stopped at different stages. The quality preset PaToH-Q, uses 3 full *V-cycles* and 3 shorter *V-cycles* as opposed to the single *V-cycle* of the default preset PaToH-D. To accelerate partitioning, both presets temporarily discard hyperedges which have more pins than some threshold. PaToH-D sets a lower threshold than PaToH-Q.

5 Experimental Evaluation

We implemented HyperFlowCutter and ReBaHFC in C++17 and compiled our code using g++8 with flags `-O3 -mtune=native -march=native`. The source code is available on GitHub¹. Experiments are performed sequentially on a cluster of Intel Xeon E5-2670 (Sandy Bridge) nodes with two Octa-Core processors clocked at 2.6 GHz with 64 GB RAM, 20 MB L3- and 8×256 KB L2-Cache, using only one core of a node.

We use the benchmark set² of Heuer and Schlag [30], which has been used to evaluate KaHyPar. It consists of 488 hypergraphs from four sources: the ISPD98 VLSI Circuit Benchmark Suite [3] (VLSI, 18 hypergraphs), the DAC 2012 Routability-Driven Placement Benchmark Suite [39] (DAC, 10), the SuiteSparse Matrix Collection [17] (SPM, 184) and the international SAT Competition 2014 [8] (Literal, Primal, Dual, 92 hypergraphs each). The set contains 173 disconnected hypergraphs, in particular all DAC instances are disconnected. Refer to [30] for more information on how the hypergraphs were derived. Unless mentioned otherwise, experiments are performed on the full benchmark set.

In the following we describe the configuration of ReBaHFC and plain HyperFlowCutter, before comparing them to competing algorithms in Section 5.4.

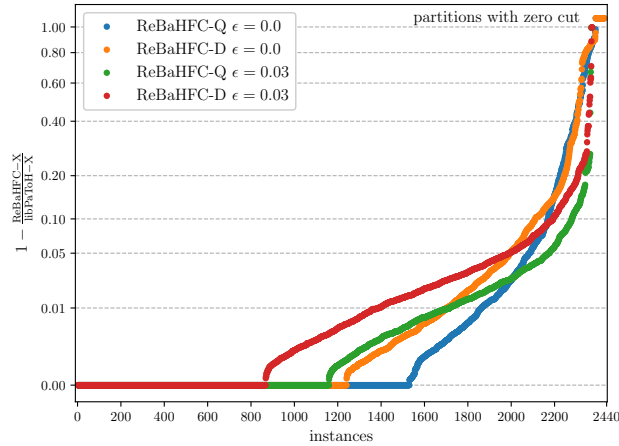
5.1 General HyperFlowCutter Configuration

To improve the solution quality of HyperFlowCutter, we run it $q \in \mathbb{N}$ times with different terminal pairs and take the minimum cut. To improve running time we run them simultaneously, in an *interleaved* fashion, as already described in [27], so that the output-sensitive running time depends on the smallest found ε -balanced cut, not the largest. We always schedule the terminal pair with the currently smallest cut to progress.

Table 1 shows the average and some quantile speedups when interleaving the execution of 20 random terminal vertex pairs, instead of running them one after another; repeated for 5 random seeds. Because consecutive execution exhibits more memory locality, we also tested a hybrid strategy where the instance with the currently smallest cut is allowed to make multiple progress iterations. Interleaving outperforms consecutive execution by a factor of 3.88 on average and also consistently beats hybrid execution. This shows that saving work is more important than memory locality. These numbers stem from a preliminary experiment

¹ Source code available at <https://github.com/kit-algo/HyperFlowCutter>

² Benchmark set and detailed statistics available at <https://algo2.itl.kit.edu/schlag/sea2017/>



■ **Figure 2** Improvement ratios $1 - \frac{\text{cut}(\text{ReBaHFC})}{\text{cut}(\text{PaToH})}$ of the two ReBaHFC variants and $\varepsilon = 0, 0.03$ compared to their initial partitions. The curves are independent from one another. Higher values are better. Ratios of zero mean ReBaHFC did not improve the initial partition. We count partitions with zero cut as an improvement ratio greater than 1, since ReBaHFC does not invoke PaToH in this case.

on a 139 hypergraph subset of the full benchmark set. The subset contains 78 sparse matrices, 17 Primal, 23 Dual, 6 Literal SAT instances, 15 VLSI and 0 DAC instances. It contains only connected hypergraphs (all DAC instances are disconnected) in order to measure the impact of interleaving, not the setup overhead for many small connected components.

5.2 ReBaHFC Configuration

We now discuss the configuration for ReBaHFC. The imbalance for the initial partition is set to the same value as the desired imbalance ε for the output partition, which proved superior to larger imbalances on initial partitions. The block-size threshold parameter α should depend on ε , so we settled on $\alpha = 0.4$ for $\varepsilon = 0.03$ and $\alpha = 0.46$ for $\varepsilon = 0$. In the TR [26] we conduct a thorough parameter study for these choices. We resize the blocks once and run HyperFlowCutter five times, interleaved as described in the previous section. This number seems to provide decent quality without increasing running time too much. We consider two variants: ReBaHFC-D, which uses PaToH with default preset and ReBaHFC-Q, which uses PaToH with quality preset.

Figure 2 shows how much ReBaHFC improves the initial partition. We run the two ReBaHFC variants for $\varepsilon = 0, 0.03$ on all hypergraphs of the benchmark set with five different random seeds and plot the ratio $1 - \frac{\text{cut}(\text{ReBaHFC})}{\text{cut}(\text{PaToH})}$ per run. Note that there is no comparison between the curves, and higher values are better for ReBaHFC. Table 2 reports how often ReBaHFC improves the initial partition, for different hypergraph classes. As expected, ReBaHFC-Q could improve fewer solutions than ReBaHFC-D since the PaToH baseline is already better. Furthermore, ReBaHFC has more opportunities for refinement with $\varepsilon = 0.03$, in particular on the DAC and VLSI instances, whereas it struggles with the Primal and Literal SAT instances for $\varepsilon = 0$. Note that other refinement algorithms do not always improve solutions either. In particular, local moving based refinement algorithms struggle with zero-gain moves in the presence of large hyperedges, and the flow-based refinement in KaHyPar can yield unbalanced solutions or reproduce the existing solution. These results show that HyperFlowCutter is a promising candidate for a refinement algorithm integrated in a multilevel partitioner, which is a direction we hope to investigate in future work.

■ **Table 2** Overview by hypergraph class, how often ReBaHFC improves the initial partition.

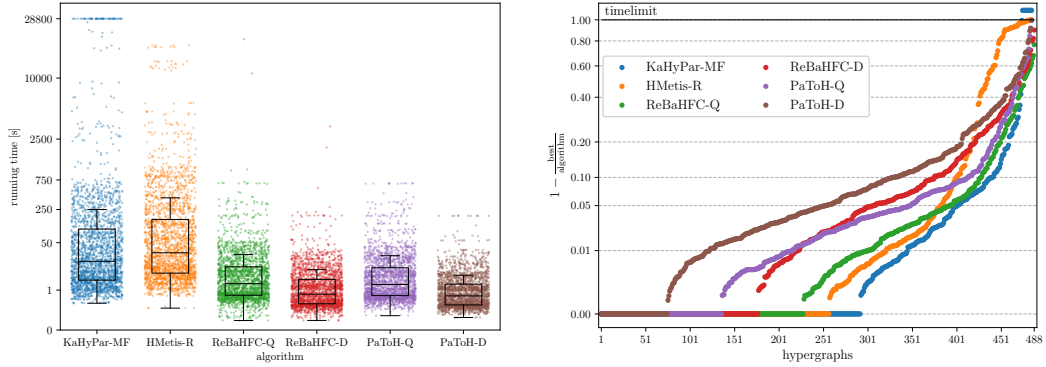
Algorithm	ε	all	SPM	Dual	Primal	Lit	DAC	VLSI
ReBaHFC-Q	0.00	37.3	39.8	47.4	23.9	33.0	66.0	34.4
ReBaHFC-D	0.00	49.2	58.4	59.8	27.4	41.3	58.0	47.8
ReBaHFC-Q	0.03	52.5	47.2	51.5	50.9	57.8	82.0	76.7
ReBaHFC-D	0.03	64.5	61.5	65.9	56.7	71.1	76.0	86.7

The PaToH runs in the experiments from Section 5.4 use other random seeds than those used internally in ReBaHFC. This makes sure that stand-alone PaToH can find smaller cuts than ReBaHFC.

5.3 Plain HyperFlowCutter Configuration

For the experiments on perfectly balanced partitioning we run plain HyperFlowCutter with up to $q = 100$ terminal pairs and take the minimum cut. This value was used already for FlowCutter [27]. With plain HyperFlowCutter we want to push the envelope on solution quality for $\varepsilon = 0$, regardless of running time – because, as the experiments show, ReBaHFC already provides a good time-quality trade-off. The most simple method for choosing starting terminals is to select random vertices. We unsuccessfully experimented with *pseudo-peripheral* terminals, i. e. two vertices that are intuitively far away from each other and at the boundary of the hypergraph. Instead we propose a selection method based on ensemble classification. Ensemble classification is a technique used in machine learning to build a strong classifier from multiple weak ones. We compute 10 bipartitions π_1, \dots, π_{10} with PaToH-D. Let $x \equiv y \Leftrightarrow \pi_i(x) = \pi_i(y)$ for all $i = 1, \dots, 10$ be the equivalence relation, in which two vertices are equivalent if they are in the same block for all ensemble bipartitions. An equivalence class is likely in the same block of a good bipartition and is thus suited as a terminal set. We order the equivalence classes by size in descending order and group two successive classes as one terminal pair. Generally speaking, the larger equivalence classes make for better terminal pairs. Based on experiments in the TR [26], we use 3 ensemble terminal pairs and 97 random vertex pairs. The reported running time for plain HyperFlowCutter always includes the running time for the 10 PaToH-D runs.

On 42 of the 488 hypergraphs, plain HyperFlowCutter with 100 terminal pairs exceeds the eight hour time limit. One downside of interleaving executions is that the solution is only available once all terminal pairs have been processed. Instead of interleaving all 100 executions, we run four waves of $\langle 1, 5, 14, 80 \rangle$ terminal pairs consecutively and interleave execution within waves. An improved bipartition is available after every wave, so that, even if the time limit is exceeded, a solution is available as long as the first wave has been completed. We chose wave sizes, so that completing waves four and three corresponds to 100 and 20 terminal pairs, respectively, as these values were used in [27]. The first wave consists of the first ensemble terminal pair, the second/third wave consist of 5/14 random terminal pairs and the fourth wave consists of 78 random as well as two additional ensemble terminal pairs. There are 438 hypergraphs for which the fourth wave finishes, 35 for which the third but not the fourth wave finishes, 6 for the second, 1 for the first and there are 8 hypergraphs which are partitioned with zero cut, using just the subset sum preprocessing.



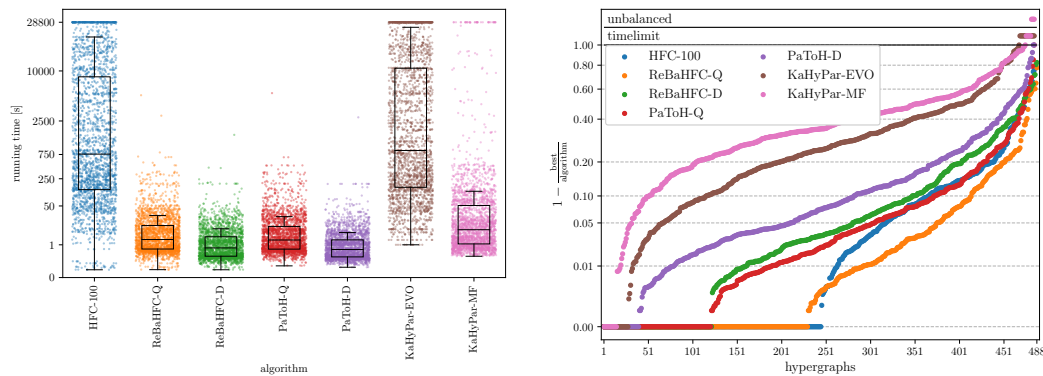
■ **Figure 3** Comparison between the algorithms for $\varepsilon = 0.03$. Left: Absolute running times for every hypergraph and random seed. Right: Performance plot relating the minimum cut per algorithm and hypergraph to the overall best cut for that hypergraph. Lower values are better.

5.4 Comparing ReBaHFC and HyperFlowCutter against State-of-the-Art Hypergraph Partitioners

In this section, we compare ReBaHFC and plain HyperFlowCutter against state-of-the-art hypergraph partitioners. After discussing our comparison methodology, we present results for two settings, namely $\varepsilon = 0.03$ and $\varepsilon = 0$.

Methodology. We run each partitioner five times with different random seeds and report the minimum cut. For every run we set a time limit of eight hours. We use the *performance plots* introduced in [45] to compare algorithms on a per-hypergraph basis regarding cut size. For each algorithm and hypergraph these plots contain a *performance ratio* $1 - \text{best}/\text{algorithm}$, which relates the minimum cut found by any algorithm to the minimum cut found by this algorithm. The ratios of each algorithm are sorted in increasing order. A ratio of 0 means that this algorithm found the smallest overall cut, the number of achieved ratios of 0 is the number of hypergraphs on which this algorithm is the best. Furthermore, algorithm A dominates algorithm B if the curve of A is strictly below that of B. We use values greater than 1 to indicate that algorithms exceeded the time limit or produced unbalanced solutions. This is clearly marked in the plots. To include partitions with zero cut, we set the performance ratio to 0, if the algorithm found the zero cut partition, and 1 otherwise. The performance plots use a cube root scaled y-axis in order to reduce right skewness [16] and to give a fine-grained view on the smaller improvements. For comparing algorithms regarding running time we use a combination of a scatter plot, which shows every measured running time, and a box plot (0.25, median, 0.75 quantiles, whiskers at most extreme points within distance $1.5 \cdot \text{IQR}$ from the upper/lower quartile). The running time plots use a fifth root scaled y-axis for a fine-grained view on areas with smaller running times, which contain more data points.

Comparison for 3% imbalance. For $\varepsilon = 0.03$ we compare ReBaHFC against the state-of-the-art hypergraph partitioning tools KaHyPar-MF (the latest version of KaHyPar with flow-based refinement) and hMETIS-R (the recursive bisection variant of hMETIS), as well as PaToH-D (default preset) and PaToH-Q (quality preset). We use the library interface of PaToH. According to the hMETIS manual, hMETIS-R is preferred over hMETIS-K (direct k-way) for bipartitions, so we exclude hMETIS-K. These tools were chosen because



■ **Figure 4** Comparison between the algorithms for $\varepsilon = 0$. Left: Absolute running times for every hypergraph and random seed. Right: performance plot relating the minimum cut per algorithm and hypergraph to the overall best cut for that hypergraph. Lower values are better.

they provide the best solution quality according to [2, 30]. We chose $\varepsilon = 0.03$ as this is a commonly used value in the literature. Plain HyperFlowCutter is excluded from this part of the experiments because it is not competitive.

Figure 3 shows the running times and a performance plot on the full benchmark set for $\varepsilon = 0.03$. In addition to the running time plot, we compare algorithms by the geometric mean of their running times. We use the geometric mean in order to give instances of different sizes a comparable influence. KaHyPar-MF finds the smallest cut on 292 hypergraphs, hMETIS-R on 257, ReBaHFC-Q on 228, ReBaHFC-D on 177, PaToH-Q on 136 and PaToH-D on 75 of the 488 hypergraphs. While KaHyPar-MF is the best algorithm regarding solution quality, it is also the slowest, exceeding the time limit on 11 hypergraphs. For the instances on which ReBaHFC-Q does not find the best solution it provides solution quality similar to hMETIS-R and only marginally worse than KaHyPar-MF. In particular its solution quality compared to the best cut deteriorates less than that of hMETIS-R. With 2.23s PaToH-Q is one order of magnitude faster than KaHyPar (34.1s) and hMETIS-R (20.1s), whereas ReBaHFC-Q (2.32s) is only slightly slower than PaToH-Q. Furthermore ReBaHFC-D (0.68s) finds more of the best solutions than PaToH-Q at a running time between PaToH-D (0.5s) and PaToH-Q. Thus ReBaHFC-Q and ReBaHFC-D provide new Pareto points in the time-quality trade-off. In the TR [26], we also report performance plots for the different hypergraph classes of the benchmark set. ReBaHFC is particularly good on the DAC and SPM instances. There are hypergraphs on which ReBaHFC is faster than PaToH. These are disconnected hypergraphs, for which ReBaHFC invokes PaToH on smaller sub-hypergraphs, due to the gap-filler optimization and the SUBSETSUM preprocessing described in Section 4.2.

Comparison for perfectly balanced partitioning. Even though the setting $\varepsilon = 0$ has received no attention in hypergraph partitioning and only some attention in graph partitioning [44, 38, 14, 9, 10, 11, 18], we consider it here. Previous studies on perfectly balanced partitioning for graphs have focused on high quality solutions through running time intensive metaheuristics such as evolutionary algorithms [44, 10, 9] or tabu search [11] and even an exact branch-and-bound algorithm [18]. Therefore, we include KaHyPar-EVO [6] (the evolutionary algorithm of KaHyPar) as well as plain HyperFlowCutter in addition to the already considered algorithms. We exclude hMETIS-R from this comparison since it rejects $\varepsilon < 0.002$ for bipartitions.

We include plain HyperFlowCutter with up to 100 terminal pairs as described in Section 5.1 and denote this configuration as HFC-100. The evolutionary algorithm KaHyPar-EVO generates, manages and improves a pool of solutions until a time limit is exceeded, and outputs the minimum cut out of all generated solutions. We set the instance-wise time limit to the maximum of the running times of HFC-100 and KaHyPar-MF to evaluate whether KaHyPar-EVO can yield better solution quality when given the same running time as HFC-100. As opposed to the original paper, we configure KaHyPar-EVO to use flow-based refinement, which further improves solution quality.

KaHyPar-MF is unable to find any balanced bipartition on 4 hypergraphs, whereas KaHyPar-EVO always finds one. Furthermore, KaHyPar-MF exceeds the time limit on 7 hypergraphs and KaHyPar-EVO on an additional 17, without reporting intermediate solutions. Figure 4 shows the running times and a performance plot of all tested algorithms. HFC-100 produces the best solutions on 245 hypergraphs, followed by ReBaHFC-Q (230), ReBaHFC-D (122), PaToH-Q (121), PaToH-D (40), KaHyPar-EVO (28) and finally KaHyPar-MF (15). This shows that with exorbitant running time, HFC-100 produces high quality solutions for $\varepsilon = 0$. However the time-quality trade-off is clearly in favor of ReBaHFC-Q, especially since the solution quality of the latter is closer to the best cut for the instances on which it does not find the best cut, as opposed to HFC-100. PaToH is better than KaHyPar for $\varepsilon = 0$ because it includes a KL [33] refinement pass as opposed to KaHyPar which only uses FM [23].

6 Conclusion

In this paper we propose and evaluate HyperFlowCutter, a hypergraph bipartitioning algorithm based on maximum flow computations. It enumerates partitions with increasing balance up to perfect balance. We also propose and evaluate ReBaHFC, a refinement algorithm based on HyperFlowCutter.

In our experimental evaluation on a large set of hypergraphs, we show that while ReBaHFC is unable to beat the state-of-the-art hypergraph partitioners in terms of quality, it is still close in terms of quality and at the same time an order of magnitude faster. Thus, it offers a new trade-off between quality and running time. For the special case of perfectly balanced bipartitioning, the plain HyperFlowCutter algorithm, while being slow, computes the highest-quality solutions. In this setting, ReBaHFC not only still beats all other partitioners but is also much faster.

In future work, it would be interesting to integrate the refinement step of ReBaHFC into multilevel partitioners to see if it can further improve their solution quality.

References

- 1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.
- 2 Yaroslav Akhremtsev, Tobias Heuer, Sebastian Schlag, and Peter Sanders. Engineering a direct k-way Hypergraph Partitioning Algorithm. In *Proceedings of the 19th Meeting on Algorithm Engineering and Experiments (ALENEX'17)*, pages 28–42. SIAM, 2017. doi:10.1137/1.9781611974768.3.
- 3 Charles J. Alpert. The ISPD98 Circuit Benchmark Suite. In *Proceedings of the 1998 International Symposium on Physical Design*, pages 80–85, 1998. doi:10.1145/274535.274546.
- 4 Charles J. Alpert, Jen-Hsin Huang, and Andrew B. Kahng. Multilevel Circuit Partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(8):655–667, August 1998. doi:10.1109/43.712098.

- 5 Charles J. Alpert and Andrew B. Kahng. Recent Directions in Netlist Partitioning: A Survey. *Integration: The VLSI Journal*, 19(1-2):1–81, 1995. doi:10.1016/0167-9260(95)00008-4.
- 6 Robin Andre, Sebastian Schlag, and Christian Schulz. Memetic Multilevel Hypergraph Partitioning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 347–354. ACM Press, July 2018. doi:10.1145/3205455.3205475.
- 7 David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner. *Graph Partitioning and Graph Clustering: 10th DIMACS Implementation Challenge*, volume 588. American Mathematical Society, 2013. doi:10.1090/conm/588.
- 8 Anton Belov, Daniel Diepold, Marijn JH Heule, and Matti Järvisalo. The SAT Competition 2014, 2014. URL: <http://www.satcompetition.org/2014/>.
- 9 Una Benlic and Jin-Kao Hao. An Effective Multilevel Memetic Algorithm for Balanced Graph Partitioning. In *IEEE International Conference on Tools with Artificial Intelligence*, pages 121–128. IEEE Computer Society, 2010. doi:10.1109/ICTAI.2010.25.
- 10 Una Benlic and Jin-Kao Hao. A Multilevel Memetic Approach for Improving Graph k-Partitions. *IEEE Transactions on Evolutionary Computation*, 15(5):624–642, October 2011. doi:10.1109/TEVC.2011.2136346.
- 11 Una Benlic and Jin-Kao Hao. An Effective Multilevel Tabu Search Approach for Balanced Graph Partitioning. *Computers & Operations Research*, 38(7):1066–1075, July 2011. doi:10.1016/j.cor.2010.10.007.
- 12 Thang Nguyen Bui and Curt Jones. Finding good approximate vertex and edge partitions is NP-hard. *Information Processing Letters*, 42(3):153–159, 1992. doi:10.1016/0020-0190(92)90140-Q.
- 13 Umit Catalyurek and Cevdet Aykanat. Hypergraph-Partitioning-Based Decomposition for Parallel Sparse-Matrix Vector Multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):673–693, 1999. doi:10.1109/71.780863.
- 14 Pierre Chardaire, Musbah Barake, and Geoff P. McKeown. A PROBE-Based Heuristic for Graph Partitioning. *IEEE Transactions on Computers*, 56(12):1707–1720, 2007. doi:10.1109/TC.2007.70760.
- 15 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.
- 16 Nicholas J. Cox. Stata Tip 96: Cube Roots. *The Stata Journal*, 11(1):149–154, 2011. doi:10.1177/1536867X1101100112.
- 17 Timothy A. Davis and Yifan Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1), 2011. doi:10.1145/2049662.2049663.
- 18 Daniel Delling and Renato F. Werneck. Better Bounds for Graph Bisection. In Leah Epstein and Paolo Ferragina, editors, *Proceedings of the 20th Annual European Symposium on Algorithms (ESA’12)*, volume 7501 of *Lecture Notes in Computer Science*, pages 407–418. Springer, 2012. doi:10.1007/978-3-642-33090-2_36.
- 19 Karen Devine, Erik Boman, Robert Heaphy, Rob Bisseling, and Umit Catalyurek. Parallel Hypergraph Partitioning for Scientific Computing. In *20th International Parallel and Distributed Processing Symposium (IPDPS’06)*. IEEE Computer Society, 2006. doi:10.1109/IPDPS.2006.1639359.
- 20 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable Contraction Hierarchies. *ACM Journal of Experimental Algorithmics*, 21(1):1.5:1–1.5:49, April 2016. doi:10.1145/2886843.
- 21 Yefim Dinitz. Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation. *Soviet Mathematics-Doklady*, 11(5):1277–1280, September 1970.
- 22 Jack Edmonds and Richard M. Karp. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *Journal of the ACM*, 19(2):248–264, April 1972. doi:10.1145/321694.321699.

- 23 Charles M. Fiduccia and Robert M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th ACM/IEEE Conference on Design Automation*, pages 175–181, 1982. doi:10.1145/800263.809204.
- 24 Lester R. Ford, Jr. and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.
- 25 Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988. doi:10.1145/48014.61051.
- 26 Lars Gottesbüren, Michael Hamann, and Dorothea Wagner. Evaluation of a Flow-Based Hypergraph Bipartitioning Algorithm. Technical report, Institute for Theoretical Informatics, Karlsruhe Institute of Technology, 2019. arXiv:1907.02053.
- 27 Michael Hamann and Ben Strasser. Graph Bisection with Pareto Optimization. *ACM Journal of Experimental Algorithmics*, 23(1):1.2:1–1.2:34, 2018. doi:10.1145/3173045.
- 28 Bruce Hendrickson and Robert Leland. A Multilevel Algorithm for Partitioning Graphs. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (SC’05)*, page 28. ACM Press, 1995. doi:10.1145/224170.224228.
- 29 Tobias Heuer, Peter Sanders, and Sebastian Schlag. Network Flow-Based Refinement for Multilevel Hypergraph Partitioning. In *Proceedings of the 17th International Symposium on Experimental Algorithms (SEA’18)*, Leibniz International Proceedings in Informatics, pages 1–19, 2018. doi:10.4230/LIPIcs.SEA.2018.1.
- 30 Tobias Heuer and Sebastian Schlag. Improving Coarsening Schemes for Hypergraph Partitioning by Exploiting Community Structure. In Costas S. Iliopoulos, Solon P. Pissis, Simon J. Puglisi, and Rajeev Raman, editors, *Proceedings of the 16th International Symposium on Experimental Algorithms (SEA’17)*, volume 75 of *Leibniz International Proceedings in Informatics*, 2017. doi:10.4230/LIPIcs.SEA.2017.21.
- 31 George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel Hypergraph Partitioning: Applications in VLSI Domain. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(1):69–79, 1999. doi:10.1109/92.748202.
- 32 George Karypis and Vipin Kumar. Multilevel K-Way Hypergraph Partitioning. In *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, 1999. doi:10.1145/309847.309954.
- 33 Brian W. Kernighan and Shen Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell System Technical Journal*, 49(2):291–307, February 1970. doi:10.1002/j.1538-7305.1970.tb01770.x.
- 34 Eugene L. Lawler. Cutsets and Partitions of hypergraphs. *Networks*, 3:275–285, 1973. doi:10.1002/net.3230030306.
- 35 Thomas Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley, 1990.
- 36 Jianmin Li, John Lillis, and C.K. Cheng. Linear decomposition algorithm for VLSI design applications. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design*, pages 223–228. IEEE Computer Society, 1995. doi:10.1109/ICCAD.1995.480016.
- 37 Huiqun Liu and D.F. Wong. Network-Flow-Based Multiway Partitioning with Area and Pin Constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(1):50–59, 1998. doi:10.1109/43.673632.
- 38 Henning Meyerhenke, Burkhard Monien, and Thomas Sauerwald. A new diffusion-based multilevel algorithm for computing graph partitions. *Journal of Parallel and Distributed Computing*, 69(9):750–761, 2009. doi:10.1016/j.jpdc.2009.04.005.
- 39 Viswanath Nagarajan, Charles J. Alpert, Cliff Sze, Zhuo Li, and Yaoguang Wei. The DAC 2012 Routability-driven Placement Contest and Benchmark Suite. In *Proceedings of the 49th Annual Design Automation Conference*, 2012. doi:10.1145/2228360.2228500.
- 40 David A. Papa and Igor L. Markov. Hypergraph Partitioning and Clustering. In Teofilo F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, 2007. doi:10.1201/9781420010749.

- 41 Jean-Claude Picard and Maurice Queyranne. On the Structure of All Minimum Cuts in a Network and Applications. *Mathematical Programming, Series A*, 22(1):121, December 1982. doi:10.1007/BF01581031.
- 42 Joachim Pistorius and Michel Minoux. An Improved Direct Labeling Method for the Max-Flow Min-Cut Computation in Large Hypergraphs and Applications. *International Transactions in Operational Research*, 10(1):1–11, 2003. doi:10.1111/1475-3995.00389.
- 43 Peter Sanders and Christian Schulz. Engineering Multilevel Graph Partitioning Algorithms. In *Proceedings of the 19th Annual European Symposium on Algorithms (ESA'11)*, volume 6942 of *Lecture Notes in Computer Science*, pages 469–480. Springer, 2011. doi:10.1007/978-3-642-23719-5_40.
- 44 Peter Sanders and Christian Schulz. Think Locally, Act Globally: Highly Balanced Graph Partitioning. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 164–175. Springer, 2013. doi:10.1007/978-3-642-38527-8_16.
- 45 Sebastian Schlag, Vitali Henne, Tobias Heuer, Henning Meyerhenke, Peter Sanders, and Christian Schulz. k-way Hypergraph Partitioning via n-Level Recursive Bisection. In *Proceedings of the 18th Meeting on Algorithm Engineering and Experiments (ALENEX'16)*, pages 53–67. SIAM, 2016. doi:10.1137/1.9781611974317.5.
- 46 Ruslan Shaydulín, Jie Chen, and Ilya Safro. Relaxation-Based Coarsening for Multilevel Hypergraph Partitioning. *Multiscale Modeling and Simulation*, 17(1):482–506, 2019. doi:10.1137/17M1152735.
- 47 A. J. Soper, Chris Walshaw, and Mark Cross. The Graph Partitioning Archive, 2004. URL: <http://chriswalshaw.co.uk/partition/>.
- 48 Aleksandar Trifunovic and William J. Knottenbelt. Parallel Multilevel Algorithms for Hypergraph Partitioning. *Journal of Parallel and Distributed Computing*, 68(5):563–581, 2008. doi:10.1016/j.jpdc.2007.11.002.
- 49 Brendan Vastenhouw and Rob Bisseling. A Two-Dimensional Data Distribution Method for Parallel Sparse Matrix-Vector Multiplication. *SIAM Review*, 47(1):67–95, 2005. doi:10.1137/S0036144502409019.
- 50 Chris Walshaw. Multilevel Refinement for Combinatorial Optimisation Problems. *Annals of Operations Research*, 131(1):325–372, October 2004. doi:10.1023/B:ANOR.0000039525.80601.15.
- 51 Hannah Honghua Yang and D.F. Wong. Efficient Network Flow Based Min-Cut Balanced Partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(12):1533–1540, 1996. doi:10.1007/978-1-4615-0292-0_41.

Parameterized Approximation Schemes for Independent Set of Rectangles and Geometric Knapsack

Fabrizio Grandoni 

IDSIA, USI-SUPSI, Manno, Switzerland
fabrizio@idsia.ch

Stefan Kratsch 

Institut für Informatik, Humboldt Universität zu Berlin, Germany
kratsch@informatik.hu-berlin.de

Andreas Wiese

Department of Industrial Engineering and Center for Mathematical Modeling,
Universidad de Chile, Chile
awiese@dii.uchile.cl

Abstract

The area of parameterized approximation seeks to combine approximation and parameterized algorithms to obtain, e.g., $(1 + \varepsilon)$ -approximations in $f(k, \varepsilon)n^{O(1)}$ time where k is some *parameter* of the input. The goal is to overcome lower bounds from either of the areas. We obtain the following results on parameterized approximability:

- In the *maximum independent set of rectangles* problem (MISR) we are given a collection of n axis parallel rectangles in the plane. Our goal is to select a maximum-cardinality subset of pairwise non-overlapping rectangles. This problem is NP-hard and also W[1]-hard [Marx, ESA'05]. The best-known polynomial-time approximation factor is $O(\log \log n)$ [Chalermsook and Chuzhoy, SODA'09] and it admits a QPTAS [Adamaszek and Wiese, FOCS'13; Chuzhoy and Ene, FOCS'16]. Here we present a *parameterized approximation scheme* (PAS) for MISR, i.e. an algorithm that, for any given constant $\varepsilon > 0$ and integer $k > 0$, in time $f(k, \varepsilon)n^{g(\varepsilon)}$, either outputs a solution of size at least $k/(1 + \varepsilon)$, or declares that the optimum solution has size less than k .
- In the (*2-dimensional*) *geometric knapsack* problem (2DK) we are given an axis-aligned square knapsack and a collection of axis-aligned rectangles in the plane (*items*). Our goal is to translate a maximum cardinality subset of items into the knapsack so that the selected items do not overlap. In the version of 2DK with rotations (2DKR), we are allowed to rotate items by 90 degrees. Both variants are NP-hard, and the best-known polynomial-time approximation factor is $2 + \varepsilon$ [Jansen and Zhang, SODA'04]. These problems admit a QPTAS for polynomially bounded item sizes [Adamaszek and Wiese, SODA'15]. We show that both variants are W[1]-hard. Furthermore, we present a PAS for 2DKR.

For all considered problems, getting time $f(k, \varepsilon)n^{O(1)}$, rather than $f(k, \varepsilon)n^{g(\varepsilon)}$, would give FPT time $f'(k)n^{O(1)}$ exact algorithms by setting $\varepsilon = 1/(k + 1)$, contradicting W[1]-hardness. Instead, for each fixed $\varepsilon > 0$, our PASs give $(1 + \varepsilon)$ -approximate solutions in FPT time.

For both MISR and 2DKR our techniques also give rise to preprocessing algorithms that take $n^{g(\varepsilon)}$ time and return a subset of at most $k^{g(\varepsilon)}$ rectangles/items that contains a solution of size at least $k/(1 + \varepsilon)$ if a solution of size k exists. This is a special case of the recently introduced notion of a polynomial-size approximate kernelization scheme [Lokshtanov et al., STOC'17].

2012 ACM Subject Classification Theory of computation → Packing and covering problems; Theory of computation → Fixed parameter tractability

Keywords and phrases parameterized approximation, parameterized intractability, independent set of rectangles, geometric knapsack

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.53



© Fabrizio Grandoni, Stefan Kratsch, and Andreas Wiese;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 53; pp. 53:1–53:16



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version A full version of the paper is available at <https://arxiv.org/abs/1906.10982>.

Funding *Fabrizio Grandoni*: Partially supported by the SNSF Excellence Grant 200020B_182865/1.
Andreas Wiese: Partially supported by the Fondecyt Regular grant 1170223.

1 Introduction

Approximation algorithms and parameterized algorithms are two well-established ways to deal with NP-hard problems. An α -approximation for an optimization problem is a polynomial-time algorithm that computes a *feasible* solution whose cost is within a factor α (that might be a function of the input size n) of the optimal cost. In particular, a *polynomial-time approximation scheme* (PTAS) is a $(1 + \varepsilon)$ -approximation algorithm running in time $n^{g(\varepsilon)}$, where $\varepsilon > 0$ is a given constant and g is some computable function. In parameterized algorithms we identify a parameter k of the input, that we informally assume to be much smaller than n . The goal here is to solve the problem optimally in *fixed-parameter tractable* (FPT) time $f(k)n^{O(1)}$, where f is some computable function. Recently, researchers started to combine the two notions (see, e.g., the survey by Marx [34]). The idea is to design approximation algorithms that run in FPT (rather than polynomial) time, e.g., to get $(1 + \varepsilon)$ -approximate solutions in time $f(k, \varepsilon)n^{O(1)}$. In this paper we continue this line of research on parameterized approximation, and apply it to two fundamental rectangle packing problems.

1.1 Our results and techniques

Our focus is on parameterized approximation algorithms. Unfortunately, as observed by Marx [34], when the parameter k is the desired solution size, computing $(1 + \varepsilon)$ -approximate solutions in time $f(k, \varepsilon)n^{O(1)}$ implies fixed-parameter tractability. Indeed, setting $\varepsilon = 1/(k+1)$ guarantees to find an optimal solution when that value equals to $k \in \mathbb{N}$ and we get time $f(k, 1/(k+1))n^{O(1)} = f'(k)n^{O(1)}$. Since the considered problems are W[1]-hard (in part, this is established in our work), they are unlikely to be FPT and similarly unlikely to have such nice approximation schemes.

Instead, we construct algorithms (for two maximization problems) that, given $\varepsilon > 0$ and an integer k , take time $f(k, \varepsilon)n^{g(\varepsilon)}$ and either return a solution of size at least $k/(1 + \varepsilon)$ or declare that the optimum is less than k . We call such an algorithm a *parameterized approximation scheme* (PAS). Note that if we run such an algorithm for each $k' \leq k$ then we can guarantee that we compute a solution with cardinality at least $\min\{k, \text{OPT}\}/(1 + \varepsilon)$ where OPT denotes the size of the optimal solution. So intuitively, for each $\varepsilon > 0$, we have an FPT-algorithm for getting a $(1 + \varepsilon)$ -approximate solution.

In this paper we consider the following two geometric packing problems, and design PASs for them.

Maximum Independent Set of Rectangles. In the *maximum independent set of rectangles* problem (MISR) we are given a set of n axis-parallel rectangles $\mathcal{R} = \{R_1, \dots, R_n\}$ in the two-dimensional plane, where R_i is the open set of points $(x_i^{(1)}, x_i^{(2)}) \times (y_i^{(1)}, y_i^{(2)})$. A feasible solution is a subset of rectangles $\mathcal{R}' \subseteq \mathcal{R}$ such that for any two rectangles $R, R' \in \mathcal{R}'$ we have $R \cap R' = \emptyset$. Our objective is to find a feasible solution of maximum cardinality $|\mathcal{R}'|$. W.l.o.g. we assume that $x_i^{(1)}, y_i^{(1)}, x_i^{(2)}, y_i^{(2)} \in \{0, \dots, 2n - 1\}$ for each $R_i \in \mathcal{R}$ (see e.g. [1]).

MISR is very well-studied in the area of approximation algorithms. The problem is known to be NP-hard [24], and the current best polynomial-time approximation factor is $O(\log \log n)$ for the cardinality case [11] (addressed in this paper), and $O(\log n / \log \log n)$ for the natural generalization with rectangle weights [12]. The cardinality case also admits a

$(1 + \varepsilon)$ -approximation with a running time of $n^{\text{poly}(\log \log(n/\varepsilon))}$ [15] and there is a (slower) QPTAS known for the weighted case [1]. The problem is also known to be $W[1]$ -hard w.r.t. the number k of rectangles in the solution [33], and thus unlikely to be solvable in FPT time $f(k)n^{O(1)}$.

In this paper we achieve the following main result:

► **Theorem 1.** *There is a PAS for MISR with running time $k^{O(k/\varepsilon^8)}n^{O(1/\varepsilon^8)}$.*

In order to achieve the above result, we combine several ideas. Our starting point is a polynomial-time construction of a $k \times k$ grid such that each rectangle in the input contains some crossing point of this grid (or we find a solution of size k directly). By applying (in a non-trivial way) a result by Frederickson [21] on planar graphs, and losing a small factor in the approximation, we define a decomposition of our grid into a collection of disjoint *groups* of cells. Each such group defines an independent instance of the problem, consisting of the rectangles strictly contained in the considered group of cells. Furthermore, we guarantee that each group spans only a *constant number* $O_\varepsilon(1)$ of rectangles of the optimum solution. Therefore in FPT time we can guess the correct decomposition, and solve each corresponding subproblem in $n^{O_\varepsilon(1)}$ time. We remark that our approach deviates substantially from prior work, and might be useful for other related problems.

An adaptation of our construction also leads to the following $(1 + \varepsilon)$ -approximative kernelization.

► **Theorem 2.** *There is an algorithm for MISR that, given $k \in \mathbb{N}$, computes in time $n^{O(1/\varepsilon^8)}$ a subset of the input rectangles of size $k^{O(1/\varepsilon^8)}$ that contains a solution of size at least $k/(1 + \varepsilon)$, assuming that the input instance admits a solution of size at least k .*

Similarly as for a PAS, if we run the above algorithm for each $k' \leq k$ we obtain a set of size $k^{O(1/\varepsilon^8)}$ that contains a solution of size at least $\min\{k, \text{OPT}\}/(1 + \varepsilon)$. Observe that any c -approximate solution on the obtained set of rectangles is also a feasible, and $c(1 + \varepsilon)$ -approximate, solution for the original instance if $\text{OPT} \leq k$ and otherwise has size at least $k/(c(1 + \varepsilon))$. Thus, our result is a special case of a *polynomial-size approximate kernelization scheme* (PSAKS) as defined in [32].¹

2-Dimensional Geometric Knapsack. In the (*2-Dimensional*) *Geometric Knapsack* problem (2DK) we are given a square *knapsack* $[0, N] \times [0, N]$, $N \in \mathbb{N}$, and a set of n items I , where each item $i \in I$ is an open rectangle $(0, w_i) \times (0, h_i)$, $N \geq w_i, h_i \in \mathbb{N}$. The goal is to find a feasible *packing* of a subset $I' \subseteq I$ of the items of maximum cardinality $|I'|$. Such packing maps each item $i \in I'$ into a new translated rectangle $(a_i, a_i + w_i) \times (b_i, b_i + h_i)$ ², so that the translated rectangles are fully contained in the knapsack and do not overlap with each other. Here we also consider a variant of 2DK *with rotations* (2DKR) where we can rotate each input rectangle by 90 degrees.

Both 2DK and 2DKR are NP-hard [31] and admit a polynomial-time $(2 + \varepsilon)$ -approximation for any constant $\varepsilon > 0$ [28]. These problems admit a QPTAS if $N = n^{O(1)}$ [2]. Somewhat surprisingly, these problems are not known to be $W[1]$ -hard when parameterized by the output number k of items. Note that showing $W[1]$ -hardness is important in our case to motivate the search for a PAS.

¹ The definition due to Lokshtanov et al. [32] is not restricted to generating a small subset of the input and a dedicated solution lifting algorithm may be used.

² Intuitively, i is shifted by a_i to the right and by b_i to the top.

► **Theorem 3.** *2DK and 2DKR are W[1]-hard when parameterized by k .*

The result is proved by parameterized reductions from a variant of the W[1]-hard SUBSET SUM problem, where we need to determine whether a set of m positive integers contains a k -tuple of numbers with sum equal to some given value t . The difficulty for reductions to 2DK or 2DKR is of course that rectangles may be freely selected and placed (and possibly rotated) to get a feasible packing.

We complement the W[1]-hardness result by giving a PAS for the case with rotations (2DKR) and a corresponding kernelization procedure like in Theorem 2 (which also yields a PSAKS).

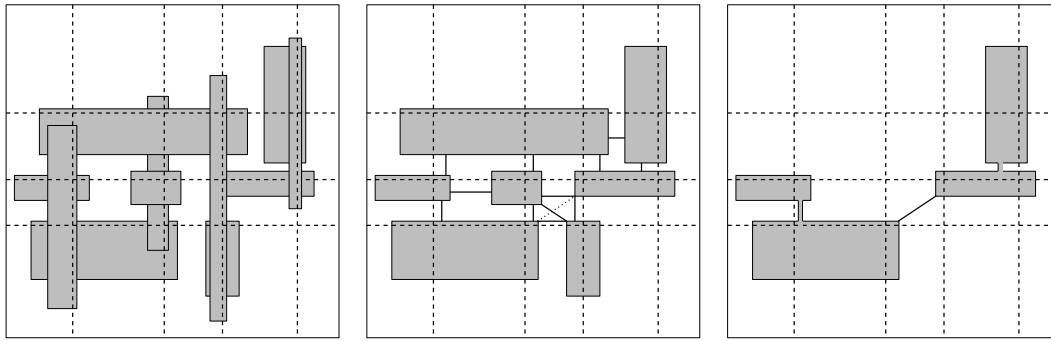
► **Theorem 4.** *For 2DKR there is a PAS with running time $k^{O(k/\epsilon)}n^{O(1/\epsilon^3)}$ and an algorithm that, given $k \in \mathbb{N}$, computes in time $n^{O(1/\epsilon^3)}$ a subset of the input items of size $k^{O(1/\epsilon)}$ that contains a solution of size at least $k/(1 + \epsilon)$, assuming that the input instance admits a solution of size at least k .*

The above result is based on a simple combination of the following two (non-trivial) building blocks: First, we show that, by losing a fraction ϵ of the items of a given solution of size k , it is possible to free a vertical strip of width $N/k^{O_\epsilon(1)}$ (unless the problem can be solved trivially). This is achieved by first sparsifying the solution using the above mentioned result by Frederickson [21]. If this is not sufficient we construct a *vertical chain* of relatively wide and tall rectangles that split the instance into a left and right side. Then we design a *resource augmentation* algorithm, however in an FPT sense: we can compute in FPT time a packing of cardinality k if we are allowed to use a knapsack where one side is enlarged by a factor $1 + 1/k^{O_\epsilon(1)}$. Note that in typical resource augmentation results the packing constraint is relaxed by a constant factor while here this amount is controlled by our parameter.

1.2 Related work

One of the first fruitful connections between parameterized complexity and approximability was observed independently by Bazgan [3] and Cesati and Trevisan [10]: They showed that EPTASs, i.e., $(1 + \epsilon)$ -approximation algorithms with $f(\epsilon)n^{O(1)}$ time, imply fixed-parameter tractability for the decision version. Thus, proofs for W[1]-hardness of the decision version became a strong tool for ruling out improvements of PTASs, with running time $n^{g(\epsilon)}$, to EPTASs. More recently, Boucher et al. [8] improved this approach by directly proving W[1]-hardness of obtaining a $(1 + \epsilon)$ -approximation, thus bypassing the requirement of a W[1]-hard decision version (see also [17]).

The systematic study of parameterized approximation as a field was initiated independently by three separate publications [9, 13, 19]. A very good introduction to the area including key definitions as well as a survey of earlier results that fit into the picture was given by Marx [34]. In particular, Marx also defined a so-called *standard FPT-approximation algorithm (with performance ratio c)* that, given input (x, k) will run for $f(k)|x|^{O(1)}$ time and return (say, for a maximization problem) a solution of value at least k/c if the optimum is at least k . As mentioned earlier, Marx pointed out that a standard FPT-approximation scheme that finds a solution of value at least $k/(1 + \epsilon)$ in time $f(k, \epsilon)|x|^{O(1)}$ if $\text{OPT} \geq k$ is not interesting to study: By setting $\epsilon = 1/(k + 1)$ we can decide the decision problem “ $\text{OPT} \geq k$?” in FPT time. Thus, such a scheme is not helpful if the decision problem is W[1]-hard and therefore unlikely to have an FPT-algorithm. Nevertheless, PASs can be useful in this case, as they imply standard FPT-approximation algorithms with ratio $1 + \epsilon$ for each fixed $\epsilon > 0$ despite W[1]-hardness.



■ **Figure 1** (Left) Dashed lines define the grid \mathcal{G} . (Middle) Rectangles from an optimal solution and the edges that form the graph G_1 . Note that in G_1 there is no edge representing the dotted connection since otherwise the graph would not be planar anymore. (Right) The graph G_2 , that captures the missing connections of G_1 .

A central goal of parameterized approximation is to settle the status of problems like DOMINATING SET or CLIQUE, which are hard to approximate and also parameterized intractable. Recently, Chen and Lin [14] made important progress by showing that DOMINATING SET admits no constant-factor approximation with running time $f(k)n^{O(1)}$ unless $\text{FPT} = \text{W}[1]$. Generally, for problems without exact FPT-algorithms, the goal is to find out whether one can beat inapproximability bounds by allowing FPT-time in some parameter; see e.g. [23, 4, 5, 6, 30, 29, 16, 22, 7]).

For the special case of MISR where all input objects are squares a PTAS is known [20] but there can be no EPTAS [33]. Recently, Galvez et al. [25] found polynomial-time algorithms for 2DK and 2DKR with approximation ratio smaller than 2 (also for the weighted case). For the special case that all input objects are squares there is a PTAS [27] and even an EPTAS [26].

2 A Parameterized Approximation Scheme for MISR

In this section we present a PAS and an approximate kernelization for MISR. We start by showing that there exists an almost optimal solution for the problem with some helpful structural properties (Sections 2.1 and 2.2). The results are then put together in Section 2.3.

2.1 Definition of the grid

We try to construct a non-uniform grid with k rows and k columns such that each input rectangle overlaps a corner of this grid (see Figure 1). To this end, we want to compute $k - 1$ vertical and $k - 1$ horizontal lines such that each input rectangle intersects one line from each set. There are instances in which our routine fails to construct such a grid (and in fact such a grid might not even exist). For such instances, we directly find a feasible solution with k rectangles and we are done.

► **Lemma 5.** *There is a polynomial time algorithm that either computes a set of at most $k - 1$ vertical lines \mathcal{L}_V with x -coordinates $\ell_1^V, \dots, \ell_{k-1}^V$ such that each input rectangle is crossed by one line in \mathcal{L}_V or computes a feasible solution with k rectangles. A symmetric statement holds for an algorithm computing a set of at most $k - 1$ horizontal lines \mathcal{L}_H with y -coordinates $\ell_1^H, \dots, \ell_{k-1}^H$.*

Proof. Let $\ell_0^V := 0$. Assume inductively that we defined the x -coordinates $\ell_0^V, \ell_1^V, \dots, \ell_{k'}^V$ such that $\ell_1^V, \dots, \ell_{k'}^V$ are the x -coordinates of the first k' constructed vertical lines. We define the x -coordinate of the $(k'+1)$ -th vertical line by $\ell_{k'+1}^V := \min_{R_i \in \mathcal{R}: x_i^{(1)} \geq \ell_{k'}^V} x_i^{(2)} - 1/2$. We continue with this construction until we reach an iteration k^* such that $\{R_i \in \mathcal{R} : x_i^{(1)} \geq \ell_{k^*-1}^V\} = \emptyset$. If $k^* \leq k$ then we constructed at most $k-1$ lines such that each input rectangle is intersected by one of these lines. Otherwise, assume that $k^* > k$. Then for each iteration $k' \in \{1, \dots, k\}$ we can find a rectangle $R_{i(k')} := \arg \min_{R_i \in \mathcal{R}: x_i^{(1)} \geq \ell_{k'-1}^V} x_i^{(2)}$. By construction, using the fact that all coordinates are integer, for any two such rectangles $R_{i(k')}, R_{i(k'')}$ with $k' \neq k''$ we have that $(x_{i(k')}^{(1)}, x_{i(k')}^{(2)}) \cap (x_{i(k'')}^{(1)}, x_{i(k'')}^{(2)}) = \emptyset$. Hence, $R_{i(k')}$ and $R_{i(k'')}$ are disjoint. Therefore, the rectangles $R_{i(1)}, \dots, R_{i(k)}$ are pairwise disjoint and thus form a feasible solution.

The algorithm for constructing the horizontal lines works symmetrically. \blacktriangleleft

We apply the algorithms due to Lemma 5. If one of them finds a set of k independent rectangles then we output them and we are done. Otherwise, we obtain the sets \mathcal{L}_V and \mathcal{L}_H . For convenience, we define two more vertical lines with x -coordinates $\ell_0^V := 0$ and $\ell_{|\mathcal{L}_V|+1}^V = 2n-1$, resp., and similarly two more horizontal lines with y -coordinates $\ell_0^H = 0$ and $\ell_{|\mathcal{L}_H|+1}^H = 2n-1$, resp.. We denote by \mathcal{G} the set of grid cells formed by these lines and the lines in $\mathcal{L}_V \cup \mathcal{L}_H$: for any two consecutive vertices lines (i.e., defined via x -coordinates ℓ_j^V, ℓ_{j+1}^V with $j \in \{0, \dots, |\mathcal{L}_V|\}$) and two consecutive horizontal grid lines (defined via y -coordinates $\ell_{j'}^H, \ell_{j'+1}^H$ with $j' \in \{0, \dots, |\mathcal{L}_H|\}$) we obtain a grid cell whose corners are the intersection of these respective lines. We interpret the grid cells as closed sets (i.e., two adjacent grid cells intersect on their boundary).

► Proposition 6. *Each input rectangle R_i contains a corner of a grid cell of \mathcal{G} . If a rectangle R intersects a grid cell g then it must contain a corner of g .*

2.2 Groups of rectangles

Let \mathcal{R}^* denote a solution to the given instance with $|\mathcal{R}^*| = k$. We prove that there is a special solution $\mathcal{R}' \subseteq \mathcal{R}^*$ of large cardinality that we can partition into $s \leq k$ groups $\mathcal{R}'_1 \dot{\cup} \dots \dot{\cup} \mathcal{R}'_s$ such that each group has constant size $O(1/\epsilon^8)$ and no grid cell can be intersected by rectangles from different groups. The remainder of this section is devoted to proving the following lemma.

► Lemma 7. *There is a constant $c = O(1/\epsilon^8)$ such that there exists a solution $\mathcal{R}' \subseteq \mathcal{R}^*$ with $|\mathcal{R}'| \geq (1-\epsilon)|\mathcal{R}^*|$ and a partition $\mathcal{R}' = \mathcal{R}'_1 \dot{\cup} \dots \dot{\cup} \mathcal{R}'_s$ with $s \leq k$ and $|\mathcal{R}'_j| \leq c$ for each j and such that if any two rectangles in \mathcal{R}' intersect the same grid cell $g \in \mathcal{G}$ then they are contained in the same set \mathcal{R}'_j .*

Given the solution \mathcal{R}^* we construct a planar graph $G_1 = (V_1, E_1)$. In V_1 we have one vertex v_i for each rectangle $R_i \in \mathcal{R}^*$. We connect two vertices $v_i, v_{i'}$ by an edge if and only if there is a grid cell $g \in \mathcal{G}$ such that R_i and $R_{i'}$ intersect g and

- R_i and $R_{i'}$ are crossed by the same horizontal or vertical line in $\mathcal{L}_V \cup \mathcal{L}_H$ or if
- R_i and $R_{i'}$ contain the top left and the bottom right corner of g , resp.

Note that we do not introduce an edge if R_i and $R_{i'}$ contain the bottom left and the top right corner of g , resp. (see Fig. 1): this way we preserve the planarity of the resulting graph, however we will have to deal with the missing connections in a later stage.

► Lemma 8. *The graph G_1 is planar.*

Next, we use a result by Frederickson [21] to obtain a subgraph G'_1 of G_1 in which each connected component has constant size.

► **Lemma 9.** *Let $\epsilon' > 0$. There exists a value $c' = O(1/(\epsilon')^2)$ such that the following holds: let $G = (V, E)$ be a planar graph. There exists a set of vertices $V' \subseteq V$ with $|V'| \geq (1 - \epsilon')|V|$ such that in the graph $G' := G[V']$ each connected component has at most c' vertices.*

Let G'_1 be the graph obtained when applying Lemma 9 to G_1 with $\epsilon' := \epsilon/2$ and let $c_1 = O((1/\epsilon)^2)$ be the respective value c' . Now we would like to claim that if two rectangles $R_i, R_{i'}$ intersect the same grid cell $g \in \mathcal{G}$ then $v_i, v_{i'}$ are in the same component of G'_1 . Unfortunately, this is not true. It might be that there is a grid cell $g \in \mathcal{G}$ such that R_i and $R_{i'}$ contain the bottom left corner and the top right corner of g , resp., and that v_i and $v_{i'}$ are in different components of G'_1 . We fix this in a second step. We define a graph $G_2 = (V_2, E_2)$. In V_2 we have one vertex for each connected component in G'_1 . We connect two vertices $w_i, w_{i'} \in V_2$ by an edge if and only if there are two rectangles $R_i, R_{i'}$ such that their corresponding vertices $v_i, v_{i'}$ in V_1 belong to the connected components of G'_1 represented by w_i and $w_{i'}$, resp., and there is a grid cell g whose bottom left and top right corner are contained in R_i and $R_{i'}$, resp.

► **Lemma 10.** *The graph G_2 is planar.*

Similarly as above, we apply Lemma 9 to G_2 with $\epsilon' := \frac{\epsilon}{2c_1}$ and let $c_2 = O((1/\epsilon')^2) = O(1/\epsilon^6)$ denote the corresponding value of c' . Denote by G'_2 the resulting graph. We define a group \mathcal{R}'_q for each connected component \mathcal{C}_q of V'_2 . The set \mathcal{R}'_q contains all rectangles R_i such that v_i is contained in a connected component C_j of G'_1 such that $w_j \in \mathcal{C}_q$. We define $\mathcal{R}' := \dot{\cup}_q \mathcal{R}'_q$.

► **Lemma 11.** *Let $R_i, R_{i'} \in \mathcal{R}'$ be rectangles that intersect the same grid cell $g \in \mathcal{G}$. Then there is a set \mathcal{R}'_q such that $\{R_i, R_{i'}\} \subseteq \mathcal{R}'_q$.*

Proof. Assume that in G_1 there is an edge connecting $v_i, v_{i'}$. Then the latter vertices are in the same connected component $C_{j'}$ of G'_1 and thus they are in the same group \mathcal{R}'_q . Otherwise, if there is no edge connecting $v_i, v_{i'}$ in G_1 then R_i and $R_{i'}$ contain the bottom left and top right corners of g , resp. Assume that v_i and $v_{i'}$ are contained in the connected components C_j and $C_{j'}$ of G'_1 , resp. Then $w_j, w_{j'} \in V'_2$, $\{w_j, w_{j'}\} \in E_2$ and $w_j, w_{j'}$ are in the same connected component of V'_2 . Hence, $R_i, R_{i'}$ are in the same group \mathcal{R}'_q . ◀

It remains to prove that each group \mathcal{R}'_q has constant size and that $|\mathcal{R}'| \geq (1 - \epsilon)|\mathcal{R}^*|$.

► **Lemma 12.** *There is a constant $c = O(1/\epsilon^8)$ such that for each group \mathcal{R}'_q it holds that $|\mathcal{R}'_q| \leq c$.*

Proof. For each group \mathcal{R}'_q there is a connected component \mathcal{C}_q of G'_2 such that \mathcal{R}'_q contains all rectangles R_i such that v_i is contained in a connected component C_j of G'_1 and $w_j \in \mathcal{C}_q$. Each connected component of G'_1 contains at most $c_1 = O(1/\epsilon^2)$ vertices of V'_1 and each component of G'_2 contains at most $c_2 = O(1/\epsilon^6)$ vertices of V'_2 . Hence, $|\mathcal{R}'_q| \leq c_1 \cdot c_2 =: c$ and $c = O((1/\epsilon^2)(1/\epsilon^6)) = O(1/\epsilon^8)$. ◀

► **Lemma 13.** *We have that $|\mathcal{R}'| \geq (1 - \epsilon)|\mathcal{R}^*|$.*

Proof. At most $\frac{\epsilon}{2} \cdot |V_1|$ vertices of G_1 are deleted when we construct G'_1 from G_1 . Each vertex in G'_1 belongs to one connected component C_j , represented by a vertex $w_j \in G_2$. At most $\frac{\epsilon}{2c_1} |V_2|$ vertices are deleted when we construct G'_2 from G_2 . These vertices represent at most $c_1 \cdot \frac{\epsilon}{2c_1} |V_2| \leq \frac{\epsilon}{2} |V'_1| \leq \frac{\epsilon}{2} |V_1|$ vertices in G_1 (and each vertex in G_1 represents one rectangle in \mathcal{R}^*). Therefore, $|\mathcal{R}'| \geq |\mathcal{R}^*| - \frac{\epsilon}{2} \cdot |V_1| - \frac{\epsilon}{2} \cdot |V_1| = (1 - \epsilon)|\mathcal{R}^*|$. ◀

This completes the proof of Lemma 7.

2.3 The algorithm

In our algorithm, we compute a solution that is at least as good as the solution \mathcal{R}' as given by Lemma 7. For each group \mathcal{R}'_j we define by \mathcal{G}_j the set of grid cells that are intersected by at least one rectangle from \mathcal{R}'_j . Since in \mathcal{R}' each grid cell can be intersected by rectangles of only one group, we have that $\mathcal{G}_j \cap \mathcal{G}_q = \emptyset$ if $j \neq q$. We want to guess the sets \mathcal{G}_j . The next lemma shows that the number of possibilities for one of those sets is polynomially bounded in k .

► **Lemma 14.** *Each \mathcal{G}_j belongs to a set \mathcal{G} of cardinality at most $k^{O(1/\varepsilon^8)}$ that can be computed in polynomial time.*

Proof. The cells \mathcal{G}_j intersected by \mathcal{R}'_j are the union of all cells $\mathcal{G}(R)$ with $R \in \mathcal{R}'_j$ where for each rectangle R the set $\mathcal{G}(R)$ denotes the cells intersected by R . Each set $\mathcal{G}(R)$ can be specified by indicating the 4 *corner* cells of $\mathcal{G}(R)$, i.e., top-left, top-right, bottom-left, and bottom-right corner. Hence there are at most k^4 choices for each such R . The claim follows since $|\mathcal{R}'_j| = O(1/\varepsilon^8)$. ◀

We hence achieve the main result of this section.

Proof of Theorem 1. Using Lemma 14, we can guess by exhaustive enumeration all the sets \mathcal{G}_j in time $k^{O(k/\varepsilon^8)}$. We obtain one independent problem for each value $j \in \{1, \dots, s\}$ which consists of all input rectangles that are contained in \mathcal{G}_j . For this subproblem, it suffices to compute a solution with at least $|\mathcal{R}'_j|$ rectangles. Since $|\mathcal{R}'_j| \leq c = O(1/\varepsilon^8)$ we can do this in time $n^{O(1/\varepsilon^8)}$ by complete enumeration. Thus, we solve each of the subproblems and output the union of the computed solutions. The overall running time is as in the claim. If all the computed solutions have size less than $(1 - \varepsilon)k$, this implies that the optimum solution is smaller than k . Otherwise we obtain a solution of size at least $(1 - \varepsilon)k \geq k/(1 + 2\varepsilon)$ and the claim follows by redefining ε appropriately. ◀

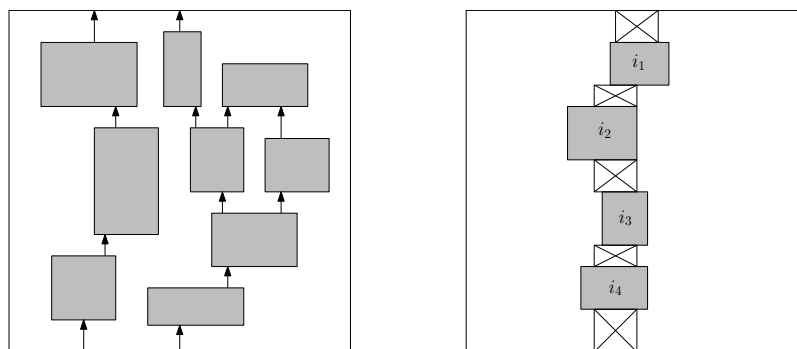
Essentially the same construction as above also gives an approximate kernelization algorithm as claimed in Theorem 2, see the full version of this work for details.

3 A Parameterized Approximation Scheme for 2DKR

In this section we present a PAS and an approximate kernelization for 2DKR. W.l.o.g., we assume that $k \geq \Omega(1/\varepsilon^3)$, since otherwise we can optimally solve the problem in time $n^{O(1/\varepsilon^3)}$ by exhaustive enumeration. In Section 3.1 we show that, if a solution of size k exists, there is a solution of size at least $(1 - \varepsilon)k$ in which no item intersects some horizontal strip $(0, N) \times (0, (1/k)^{O(1/\varepsilon)}N)$ at the bottom of the knapsack. In Section 3.2 we show that, if there exists a solution of size k' that does not use the mentioned strip, then we can compute in polynomial time a set of size $(k')^{O(1/\varepsilon)}$ that contains a solution of size k' (where we are allowed to use the full knapsack). Combining these two results gives Theorem 4.

3.1 Freeing a Horizontal Strip

In this section, we prove the following lemma that shows the existence of a near-optimal solution that leaves a sufficiently tall empty horizontal strip in the knapsack (assuming $k \geq \Omega(1/\varepsilon^3)$). W.l.o.g., $\varepsilon \leq 1$. Since we can rotate the items by 90 degrees, we can assume w.l.o.g. that $w_i \geq h_i$ for each item $i \in I$.



■ **Figure 2** The left figure shows the arcs of the graph G . Each item corresponds to one vertex of the graph. The right figure shows the items i_1, \dots, i_K and the deletion rectangles between them.

► **Lemma 15.** *Let $k \in \mathbb{N}$, $k = \Omega(1/\epsilon^3)$, and $\epsilon > 0$. Given an instance of 2DKR with a solution of size k , there exists a solution of size at least $(1 - \epsilon)k$ in which no packed item intersects $(0, N) \times (0, (1/k)^c N)$, for a proper constant $c = O(1/\epsilon)$.*

We classify items into large and thin items. Via a shifting argument, we get the following lemma.

► **Lemma 16.** *There is an integer $B \in \{1, \dots, \lceil 8/\epsilon \rceil\}$ such that by losing a factor of $1 + \epsilon$ in the objective we can assume that the input items are partitioned into*

- large items L such that $h_i \geq (1/k)^B N$ (and thus also $w_i \geq (1/k)^B N$) for each item $i \in L$,
- thin items T such that $h_i < (1/k)^{B+2} N$ for each item $i \in T$.

Let B be the integer due to Lemma 16 and we work with the resulting item classification. If $|T| \geq k$ then we can create a solution of size k satisfying the claim of Lemma 15 by simply stacking k thin items on top of each other: any k thin items have a total height of at most $k \cdot (1/k)^{B+2} N \leq (1/k)^2 N$. Thus, from now on assume that $|T| < k$.

Sparsifying large items. Our strategy is now to delete some of the large items and move the remaining items. This will allow us to free the area $[0, N] \times [0, (1/k)^{O(1/\epsilon)} N]$ of the knapsack. Denote by OPT' the almost optimal solution obtained by applying Lemma 16. We remove the items in $\text{OPT}'_T := \text{OPT}' \cap T$ temporarily; we will add them back later.

We construct a directed graph $G = (V, A)$ where we have one vertex $v_i \in V$ for each item $i \in \text{OPT}'_L := \text{OPT}' \cap L$. We connect two vertices $v_i, v_{i'}$ by an arc $a = (v_i, v_{i'})$ if and only if we can draw a vertical line segment of length at most $(1/k)^B N$ that connects item i with item i' without intersecting any other item such that i' lies above i , i.e., the bottom coordinate of i' is at least as large as the top coordinate of i , see Figure 2 for a sketch. We obtain the following proposition since for each edge we can draw a vertical line segment and these segments do not intersect each other.

► **Proposition 17.** *The graph G is planar.*

Next, we apply Lemma 9 to G with $\epsilon' := \epsilon$. Let $G' = (V', A')$ be the resulting graph. We remove from OPT'_L all items $i \in V \setminus V'$ and denote by OPT''_L the resulting solution. We push up all items in OPT''_L as much as possible. If now the strip $(0, N) \times (0, (1/k)^B N)$ is not intersected by any item then we can place all the items in T into the remaining space. Their total height can be at most $k \cdot (1/k)^{B+2} N \leq (1/k)^{B+1} N$ and thus we can leave a strip of height $(1/k)^B N - (1/k)^{B+1} N \geq (1/k)^{O(1/\epsilon)} N$ and width N empty. This completes the proof of Lemma 15 for this case.

Assume next that the strip $(0, N) \times (0, (1/k)^B N)$ is intersected by some item: the following lemma implies that there is a set of $c' = O(1/\epsilon^2)$ vertices whose items intuitively connect the top and the bottom edge of the knapsack.

► **Lemma 18.** *Assume that in OPT_L'' there is an item i_1 intersecting $(0, N) \times (0, (1/k)^B N)$. Then G contains a path $v_{i_1}, v_{i_2}, \dots, v_{i_K}$ with $K \leq c' = O(1/\epsilon^2)$, such that the distance between i_K and the top edge of the knapsack is less than $(1/k)^B N$.*

Proof. Let C denote all vertices v in G' such that there is a directed path from v_{i_1} to v in G' . The vertices in C are contained in the connected component C' in G' that contains v_{i_1} . Note that $|C| \leq |C'| \leq c'$. We claim that C must contain a vertex v_j whose corresponding item j is closer than $(1/k)^B N$ to the top edge of the knapsack. Otherwise, we would have been able to push up all items corresponding to vertices in C by $(1/k)^B N$ units: first we could have pushed up all items such that their corresponding vertices have no outgoing arc, then all items such that their vertices have outgoing arcs pointing at the former set of vertices, and so on. By definition of C , there must be a path connecting v_{i_1} with v_j . This path $v_{i_1}, v_{i_2}, \dots, v_{i_K} = v_j$ contains only vertices in C and hence its length is bounded by c' . The claim follows. ◀

Our goal is now to remove the items i_1, \dots, i_K due to Lemma 18 and $O(K) = O(1/\epsilon^2)$ more large items from OPT_L'' . Since we can assume that $k \geq \Omega(1/\epsilon^3)$ this will lose only a factor of $1 + O(\epsilon)$ in the objective. To this end we define $K + 1$ *deletion rectangles*, see Figure 2. We place one such rectangle R_ℓ between any two consecutive items $i_\ell, i_{\ell+1}$. The height of R_ℓ equals the vertical distance between i_ℓ and $i_{\ell+1}$ (at most $(1/k)^B N$) and the width of R_ℓ equals $(1/k)^B N$. Since $v_{i_\ell}, v_{i_{\ell+1}}$ are connected by an arc in G' , we can draw a vertical line segment connecting i_ℓ with $i_{\ell+1}$. We place R_ℓ such that it is intersected by this line segment. Note that for the horizontal position of R_ℓ there are still several possibilities and we choose one arbitrarily. Finally, we place a special deletion rectangle between the item i_K and the top edge of the knapsack and another special deletion rectangle between the item i_1 and the bottom edge of the knapsack. The heights of these rectangles equal the distance of i_1 and i_K with the bottom and top edge of the knapsack, resp. (which is at most $(1/k)^B N$), and their widths equal $(1/k)^B N$. They are placed such that they touch the bottom edge of i_1 and the top edge of i_K , resp.

► **Lemma 19.** *Each deletion rectangle can intersect at most 4 large items in its interior. Hence, there can be only $O(K) \leq O(c') = O(1/\epsilon^2)$ large items intersecting a deletion rectangle in their interior.*

Observe that the deletion rectangles and the items in $\{i_1, \dots, i_K\}$ separate the knapsack into a left and a right part with items OPT_{left}'' and OPT_{right}'' , resp. We delete all items in i_1, \dots, i_K and all items intersecting the interior of a deletion rectangle. Each deletion rectangle and each item in $\{i_1, \dots, i_K\}$ has a width of at least $(1/k)^B N$. Thus, we can move all items in OPT_{left}'' simultaneously by $(1/k)^B N$ units to the right. After this, no large item intersects the area $(0, (1/k)^B N) \times (0, N)$. We rotate the resulting solution by 90 degrees, hence getting an empty horizontal strip $(0, N) \times (0, (1/k)^B N)$. The total height of items in OPT_T' is at most $k \cdot (1/k)^{B+2} N \leq (1/k)^{B+1} N$. Therefore, the items in OPT_T' can be stacked (one on top of the other) inside a horizontal strip of height $(1/k)^{B+1} N$ that can be placed right below the rectangles in $\text{OPT}_{left}'' \cup \text{OPT}_{right}''$. This leaves an empty horizontal strip of height $(1/k)^B N - (1/k)^{B+1} N \geq (1/k)^{O(1/\epsilon)} N$ at the bottom of the knapsack. This completes the proof of Lemma 15.

3.2 FPT-algorithm with resource augmentation

We now compute a packing that contains as many items as the solution due to Lemma 15. However, it might use the space of the entire knapsack. In particular, we use the free space in the knapsack in the latter solution in order to round the sizes of the items. In the following lemma the reader may think of $k' = (1 - \epsilon)k$ and $\tilde{k} = k^{O(1/\epsilon)}$.

► **Lemma 20.** *Let $k', \tilde{k} \in \mathbb{N}$. There is an algorithm for 2DKR with a running time of $(\tilde{k}k')^{O(k')}n^{O(1)}$ that computes a solution of size k' or asserts that there is no solution of size k' fitting into a restricted knapsack $[0, N] \times [0, (1 - 1/\tilde{k})N]$. Also, in time $n^{O(1)}$ we can compute a set of size $O(\tilde{k}(k')^2)$ that contains a solution of size k' if there is such a solution that fits into the latter knapsack.*

Note that Lemma 20 yields an FPT algorithm if we are allowed to increase the size of the knapsack by a factor $1 + O(1/\tilde{k})$ where \tilde{k} is a second parameter.

In the remainder of this section, we prove Lemma 20 and we do not differentiate between large and thin items anymore. Assume that there exists a solution OPT'' of size k' that leaves the area $[0, N] \times [0, N/\tilde{k}]$ of the knapsack empty. We want to compute a solution of size k' . We use the empty space in order to round the heights of the items in the packing of OPT'' to integral multiples of $N/(k'\tilde{k})$. Note that in OPT'' an item i might be rotated. Thus, depending on this we actually want to round its height h_i or its width w_i . To this end, we define rounded heights and widths by $\hat{h}_i := \left\lceil \frac{h_i}{N/(k'\tilde{k})} \right\rceil N/(k'\tilde{k})$ and $\hat{w}_i := \left\lceil \frac{w_i}{N/(k'\tilde{k})} \right\rceil N/(k'\tilde{k})$ for each item i .

► **Lemma 21.** *There exists a feasible packing for all items in OPT'' even if for each rotated item i we increase its width w_i to \hat{w}_i and for each non-rotated item $i' \in \text{OPT}''$ we increase its height $h_{i'}$ to $\hat{h}_{i'}$.*

To visualize the packing due to Lemma 21 one might imagine a container of height \hat{h}_i and width w_i for each non-rotated item i and a container of height $h_{i'}$ and width $\hat{w}_{i'}$ for each rotated item i' . Next, we group the items according to their values \hat{h}_i and \hat{w}_i . We define $I_h^{(j)} := \{i \in I \mid \hat{h}_i = jN/(k'\tilde{k})\}$ and $I_w^{(j)} := \{i \in I \mid \hat{w}_i = jN/(k'\tilde{k})\}$ for each $j \in \{1, \dots, k'\tilde{k}\}$. The crucial observation is now that from each set $I_h^{(j)}$ it suffices to consider only the k' items with smallest width. If OPT'' uses an item from $I_h^{(j)}$ with larger width then we can replace it by one of the k' thinner items that is not contained in OPT'' . A symmetric statement holds for the sets $I_w^{(j)}$.

► **Lemma 22.** *We can assume that from each set $I_h^{(j)}$ the solution OPT'' contains only items among the k' items in $I_h^{(j)}$ with smallest width. Similarly, from each set $I_w^{(j)}$ the solution OPT'' contains only items among the k' items in $I_w^{(j)}$ with smallest height.*

We eliminate from each set $L_h^{(j)}$ and $L_w^{(j)}$ the items that are not among the k' items with smallest width and height, resp. At most $2k' \cdot k'\tilde{k} = O(\tilde{k}(k')^2)$ items remain, denote them by \bar{I} . Then, in time $(\tilde{k}k')^{O(k')}$ we can solve the remaining problem by completely enumerating over all subsets of \bar{I} with at most k' elements. For each enumerated set we check within the given time bounds whether its items can be packed into the knapsack (possibly via rotating some of them) by guessing sufficient auxiliary information. Therefore, if a solution of size k' for a knapsack of width N and height $(1 - 1/\tilde{k})N$ exists, then we will find a solution of size k' that fits into a knapsack of width and height N .

Now the proof of Theorem 4 follows by using Lemma 15 and then applying Lemma 20 with $k' = (1 - \epsilon)k$ and $\tilde{k} = k^{O(1/\epsilon)}$. The set \bar{I} is the claimed set (which intuitively forms the approximative kernel), we compute a solution of size at least $(1 - \epsilon)k \geq k/(1 + 2\epsilon)$ and we can redefine ϵ appropriately.

4 Hardness of Geometric Knapsack

We show that 2DK and 2DKR are both $W[1]$ -hard for parameter k by reducing from a variant of SUBSET SUM. Recall that in SUBSET SUM we are given m positive integers x_1, \dots, x_m as well as integers t and k , and have to determine whether some k -tuple of the numbers sums to t ; this is $W[1]$ -hard with respect to k [18]. In the variant MULTI-SUBSET SUM it is allowed to choose numbers more than once. It is easy to verify that the proof for $W[1]$ -hardness of SUBSET SUM due to Downey and Fellows [18] extends also to MULTI-SUBSET SUM. In our reduction to 2DKR we prove that rotations are not required for optimal solutions, making $W[1]$ -hardness of 2DK a free consequence.

Proof sketch for Theorem 3. We give a polynomial-time parameterized reduction from MULTI-SUBSET SUM to 2DKR with output parameter $k' = O(k^2)$; this establishes $W[1]$ -hardness of 2DKR.

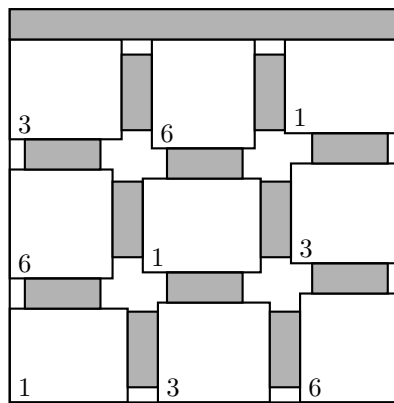
Observe that, for any packing of items into the knapsack, there is an upper bound of N on the total width of items that intersect any horizontal line through the knapsack, and similarly an upper bound of N for the total height of items along any vertical line. We will let the dimensions of some items depend on numbers x_i from the input instance (x_1, \dots, x_m, t, k) of MULTI-SUBSET SUM such that, using these upper bound inequalities, a correct packing certifies that $y_1 + \dots + y_k = t$ for some k of the numbers. The key difficulty is that there is a lot of freedom in the choice of which items to pack and where in case of a no instance.

To deal with this, the items corresponding to numbers x_i from the input are all *almost* squares and their dimensions are incomparable. Concretely, an item corresponding to some number x_i has height $L + S + x_i$ and width $L + S + 2t - x_i$; we call such an item a *tile*. (The exact values of L and S are immaterial here, but $L \gg S \gg t > x_i$ holds.) Thus, when using, e.g., a tile of smaller width (i.e., smaller value of x_i) it will occupy “more height” in the packing. The knapsack is only slightly larger than a k by k grid of such tiles, implying that there is little freedom for the placement. Let us also assume for the moment, that no rotations are used.

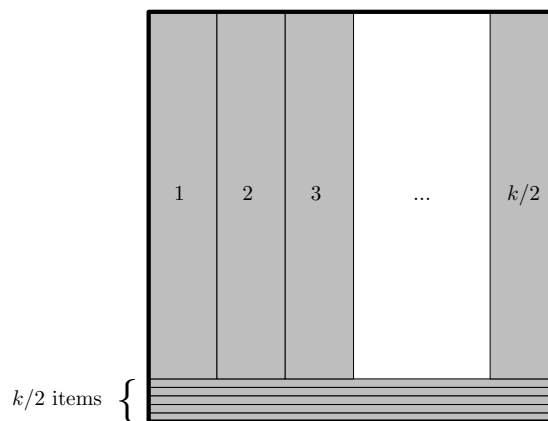
Accordingly, we can specify k vertical lines that are guaranteed to intersect all tiles of any packing that uses k^2 tiles, by using pairwise distance $L - 1$ between them. Moreover, each line is intersecting exactly k private tiles. The same holds for a similar set of k horizontal lines. Together we get an upper bound of N for the sum of the widths (heights) along any horizontal (vertical) line. Since the numbers x_i occur negatively in widths, we effectively get lower bounds for them from the horizontal lines. When the sizes of these tiles (and the auxiliary items below) are appropriately chosen, it follows that all upper bound equalities must be tight. This in turn, due to the exact choice of N , implies that there are k numbers y_1, \dots, y_k with sum equal to t .

Unsurprisingly, using just the tiles we cannot guarantee that a packing exists when given a yes-instance. This can be fixed by adding a small number of flat/thin items that can be inserted between the tiles (see Figure 3, but note that it does not match the size ratios from this proof); these have dimension $L \times S$ or $S \times L$. Because one dimension of these items is large (namely L) they must be intersected by the above horizontal or vertical lines. Thus, they can be proved to enter the above inequalities in a uniform way, so that the proof idea goes through.

Finally, let us address the question of why we can assume that there are no rotations. This is achieved by letting the width of any tile be larger than the height of any tile, and adding a final auxiliary item of width N and small height, called the *bar*. To get the desired number of items in a solution packing, it can be ensured that the bar must be used as no



■ **Figure 3** A sketch of the packing used in Theorem 3 for a solution with $k = 3$ and $1 + 3 + 6 = 10$. Items corresponding to the same number have the same size. The figure is not to scale: The gray items should be much flatter and the clear ones should look like squares of almost identical size.



■ **Figure 4** Example showing that Lemma 15 cannot be generalized to 2DK (without rotations). The total height of the $k/2$ items on the bottom of the knapsack can be made arbitrarily small. Suppose that we wanted to free up an area of height $f(k) \cdot N$ and width N or of height N and width $f(k) \cdot N$ (for some fixed function f). If the total height of the items on the bottom is smaller than $f(k) \cdot N$ then we would have to eliminate the $k/2$ items on the bottom or the $k/2$ items on top. Thus, we would lose a factor of $2 > 1 + \varepsilon$ in the approximation ratio.

more than k^2 tiles can fit into $N \times N$ and there is a limited supply of flat/thin items. W.l.o.g., the bar is not rotated. It can then be checked that using at least one tile in its rotated form will violate one of the upper bounds for the height. This completes the proof sketch. ◀

5 Open Problems

This paper leaves several interesting open problems. A first obvious question is whether there exists a PAS also for 2DK (i.e., in the case without rotations). We remark that the algorithm from Lemma 20 can be easily adapted to the case without rotations. Unfortunately, Lemma 15 does not seem to generalize to the latter case. Indeed, there are instances in which we lose up to a factor of 2 if we require a strip of width $\Omega_{\varepsilon,k}(1) \cdot N$ to be emptied, see

Figure 4. We also note that both our PASs work for the cardinality version of the problems: an extension to the weighted case is desirable. Unlike related results in the literature (where extension to the weighted case follows relatively easily from the cardinality case), this seems to pose several technical issues.

We remark that all the problems considered in this paper might admit a PTAS in the standard sense, which would be a strict improvement on our PASs. Indeed, the existence of a QPTAS for these problems [1, 2, 15] suggests that such PTASs are likely to exist. However, finding those PTASs is a very well-known and long-standing problem in the area. We hope that our results can help to achieve this challenging goal.

References

- 1 Anna Adamaszek and Andreas Wiese. Approximation Schemes for Maximum Weight Independent Set of Rectangles. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2013)*, pages 400–409. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.50.
- 2 Anna Adamaszek and Andreas Wiese. A quasi-PTAS for the Two-dimensional Geometric Knapsack Problem. In *Proceedings of the Twenty-sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 1491–1505. SIAM, 2015. URL: <http://dl.acm.org/citation.cfm?id=2722129.2722227>.
- 3 Cristina Bazgan. Schémas d’approximation et Complexité Paramétrée, Rapport du stage (DEA). Technical report, Université Paris Sud, 1995.
- 4 Cristina Bazgan, Morgan Chopin, André Nichterlein, and Florian Sikora. Parameterized approximability of maximizing the spread of influence in networks. *J. Discrete Algorithms*, 27:54–65, 2014. doi:10.1016/j.jda.2014.05.001.
- 5 Cristina Bazgan, Morgan Chopin, André Nichterlein, and Florian Sikora. Parameterized Inapproximability of Target Set Selection and Generalizations. *Computability*, 3(2):135–145, 2014. doi:10.3233/COM-140030.
- 6 Cristina Bazgan and André Nichterlein. Parameterized Inapproximability of Degree Anonymization. In *Proceedings of the 9th International Symposium on Parameterized and Exact Computation (IWPEC 2014)*, pages 75–84, 2014. doi:10.1007/978-3-319-13524-3_7.
- 7 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A c^kn 5-Approximation Algorithm for Treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016. doi:10.1137/130947374.
- 8 Christina Boucher, Christine Lo, and Daniel Lokshantov. Consensus Patterns (Probably) Has no EPTAS. In *Proceedings of the 23rd Annual European Symposium (ESA 2015)*, pages 239–250, 2015. doi:10.1007/978-3-662-48350-3_21.
- 9 Liming Cai and Xiuzhen Huang. Fixed-Parameter Approximation: Conceptual Framework and Approximability Results. In *Parameterized and Exact Computation (IWPEC 2006)*, pages 96–108, 2006. doi:10.1007/11847250_9.
- 10 Marco Cesati and Luca Trevisan. On the Efficiency of Polynomial Time Approximation Schemes. *Inf. Process. Lett.*, 64(4):165–171, 1997. doi:10.1016/S0020-0190(97)00164-6.
- 11 P. Chalermsook and J. Chuzhoy. Maximum independent set of rectangles. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA ’09)*, pages 892–901. SIAM, 2009.
- 12 Timothy M Chan and Sarel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012.
- 13 Yijia Chen, Martin Grohe, and Magdalena Grüber. On Parameterized Approximability. In *Parameterized and Exact Computation (IWPEC 2006)*, pages 109–120, 2006. doi:10.1007/11847250_10.

- 14 Yijia Chen and Bingkai Lin. The Constant Inapproximability of the Parameterized Dominating Set Problem. In *Proceedings of the IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS 2016)*, pages 505–514, 2016. doi:10.1109/FOCS.2016.61.
- 15 Julia Chuzhoy and Alina Ene. On Approximating Maximum Independent Set of Rectangles. In *Proceedings of the IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS 2016)*, pages 820–829, 2016. doi:10.1109/FOCS.2016.92.
- 16 Vincent Cohen-Addad and Arnaud de Mesmay. A Fixed Parameter Tractable Approximation Scheme for the Optimal Cut Graph of a Surface. In *Proceedings of the 23rd Annual European Symposium (ESA 2015)*, pages 386–398, 2015. doi:10.1007/978-3-662-48350-3_33.
- 17 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Lower Bounds for Approximation Schemes for Closest String. In *15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016)*, pages 12:1–12:10, 2016. doi:10.4230/LIPIcs.SWAT.2016.12.
- 18 Rodney G. Downey and Michael R. Fellows. Fixed-Parameter Tractability and Completeness II: On Completeness for W[1]. *Theor. Comput. Sci.*, 141(1&2):109–131, 1995. doi:10.1016/0304-3975(94)00097-3.
- 19 Rodney G. Downey, Michael R. Fellows, and Catherine McCartin. Parameterized Approximation Problems. In *Proceedings of the Second International Workshop on Parameterized and Exact Computation (IWPEC 2006)*, pages 121–129, 2006. doi:10.1007/11847250_11.
- 20 Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM Journal on Computing*, 34(6):1302–1323, 2005.
- 21 Greg N Federickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, 1987.
- 22 Andreas Emil Feldmann. Fixed Parameter Approximations for k-Center Problems in Low Highway Dimension Graphs. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015)*, pages 588–600, 2015. doi:10.1007/978-3-662-47666-6_47.
- 23 Michael R. Fellows, Ariel Kulik, Frances A. Rosamond, and Hadas Shachnai. Parameterized Approximation via Fidelity Preserving Transformations. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming (ICALP 2012)*, pages 351–362, 2012. doi:10.1007/978-3-642-31594-7_30.
- 24 Robert J Fowler, Michael S Paterson, and Steven L Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information processing letters*, 12(3):133–137, 1981.
- 25 Waldo Gálvez, Fabrizio Grandoni, Sandy Heydrich, Salvatore Ingala, Arindam Khan, and Andreas Wiese. Approximating Geometric Knapsack via L-Packings. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2017)*, pages 260–271, 2017. doi:10.1109/FOCS.2017.32.
- 26 Sandy Heydrich and Andreas Wiese. Faster approximation schemes for the two-dimensional knapsack problem. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 79–98. SIAM, 2017.
- 27 Klaus Jansen and Roberto Solis-Oba. A polynomial time approximation scheme for the square packing problem. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 184–198. Springer, 2008.
- 28 Klaus Jansen and Guochaun Zhang. On rectangle packing: maximizing benefits. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 204–213. SIAM, 2004.
- 29 Sudeshna Kolay, Pranabendu Misra, M. S. Ramanujan, and Saket Saurabh. Parameterized Approximations via d-Skew-Symmetric Multicut. In *Proceedings of the 39th International Symposium (MFCS 2014)*, pages 457–468, 2014. doi:10.1007/978-3-662-44465-8_39.
- 30 Michael Lampis. Parameterized Approximation Schemes Using Graph Widths. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP 2014)*, pages 775–786, 2014. doi:10.1007/978-3-662-43948-7_64.

53:16 Parameterized Approximation Schemes

- 31 Joseph YT Leung, Tommy W Tam, Chin S Wong, Gilbert H Young, and Francis YL Chin. Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10(3):271–275, 1990.
- 32 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2017)*, pages 224–237, 2017. doi:10.1145/3055399.3055456.
- 33 Dániel Marx. Efficient Approximation Schemes for Geometric Problems? In *Algorithms - Proceedings of ESA 2005*, pages 448–459, 2005. doi:10.1007/11561071_41.
- 34 Dániel Marx. Parameterized Complexity and Approximation Algorithms. *Comput. J.*, 51(1):60–78, 2008. doi:10.1093/comjnl/bxm048.

Packing Cars into Narrow Roads: PTASs for Limited Supply Highway

Fabrizio Grandoni 

IDSIA, USI-SUPSI, Manno, Switzerland
fabrizio@idsia.ch

Andreas Wiese

Department of Industrial Engineering and Center for Mathematical Modeling,
Universidad de Chile, Chile
awiese@dii.uchile.cl

Abstract

In the *Highway* problem, we are given a path with n edges (the highway), and a set of m drivers, each one characterized by a subpath and a budget. For a given assignment of edge prices (the tolls), the highway owner collects from each driver the total price of the associated path when it does not exceed drivers's budget, and zero otherwise. The goal is to choose the prices to maximize the total profit. A PTAS is known for this (strongly NP-hard) problem [Grandoni,Rothvoss-SODA'11,SICOMP'16].

In this paper we study the *limited supply* generalization of Highway, that incorporates capacity constraints. Here the input also includes a capacity $u_e \geq 0$ for each edge e ; we need to select, among drivers that can afford the required price, a subset such that the number of drivers that use each edge e is at most u_e (and we get profit only from selected drivers). To the best of our knowledge, the only approximation algorithm known for this problem is a folklore $O(\log m)$ approximation based on a reduction to the related Unsplittable Flow on a Path problem (UFP). The main result of this paper is a PTAS for limited supply highway.

As a second contribution, we study a natural generalization of the problem where each driver i demands a different amount d_i of capacity. Using known techniques, it is not hard to derive a QPTAS for this problem. Here we present a PTAS for the case that drivers have uniform budgets. Finding a PTAS for non-uniform-demand limited supply highway is left as a challenging open problem.

2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases approximation algorithms, pricing problems, highway problem, unsplittable flow on a path

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.54

Funding *Fabrizio Grandoni*: Partially supported by the SNSF Excellence Grant 200020B_182865/1.
Andreas Wiese: Partially supported by the Fondecyt Regular grant 1170223.

1 Introduction

In the *Highway* problem we are given a path graph $G = (V, E)$ with n edges (the *highway*) and a set D of m drivers. Each driver i is characterized by a subpath P_i of G , and by a budget $B_i \in \mathbb{N}^+$. We have to fix a price $p_e \geq 0$ on each edge e (the same for all drivers). Then, for each driver i , we get a profit of $p(i) := \sum_{e \in P_i} p_e$ (i.e., the total price over the edges *used* by i), provided that $p(i) \leq B_i$, and otherwise 0. Intuitively, each driver wishes to travel along subpath P_i , but it is not going to do that if the total requested price exceeds her budget. Our goal is to choose the prices to maximize the total profit from all drivers.

It is not hard to imagine applications for this problem, besides the obvious one suggested by its name. For example, highway edges might represent links of a (high-bandwidth) telecommunication network. Alternatively, one might interpret the highway as a period of time, and the edges as time slots: now drivers are clients who need a service for a given interval of time.



© Fabrizio Grandoni and Andreas Wiese;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 54; pp. 54:1–54:14



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Highway is well studied. It was shown to be weakly NP-hard in [9] via a reduction from *Partition*, and strongly NP-hard in [20] via a reduction from *Max-2-SAT*. There is a simple $O(\log m)$ -approximation that works for much more general instances. This was improved to $O(\log n)$ in [3] using ideas in [26], and to $O(\log n / \log \log n)$ in [22]. A QPTAS for the problem was presented in [21]. Finally, a PTAS was given in [25].

In this paper we study the *Limited Supply Highway* problem (Ls-Highway), which is a natural generalization of Highway with capacity constraints. Here we are additionally given an integral capacity $u_e \in \mathbb{N}^+$ for each edge e . A solution is now given by a price $p_e \geq 0$ on each edge e plus a subset $S \subseteq D$ of drivers that satisfy the following *capacity constraint*: the total number of selected drivers that use each edge e is at most u_e , i.e. $|\{i \in S : e \in P_i\}| \leq u_e$. The profit from each driver is defined in the same way as in Highway, however now we obtain profit only from the selected drivers S . Observe that there might be drivers that can afford to pay for the considered prices and are still excluded (i.e., they cannot take the highway) due to capacity constraints. Capacity constraints make sense in some of the mentioned applications, e.g., optimal networks might have insufficient bandwidth to accommodate all candidate users and the authority handling the network could exclude some of these users (regardless of their budget). The same argument applies to a company selling a limited resource, such as computational power, over time slots. The best known approximation for Ls-Highway is, to the best of our knowledge, a folklore $O(\log m)$ approximation based on a reduction to the related *Unsplittable Flow on Path* problem (UFP). Details about this reduction are given later.

In this paper we also consider a *non-uniform demand* generalization of Ls-Highway, next denoted as NuLs-Highway, where each driver i has a demand $d_i \in \mathbb{N}$. W.l.o.g., we can assume that $d_i \leq \min_{e \in P_i} \{u_e\}$ (otherwise driver i can be discarded). Now the subset S of selected drivers has to satisfy $\sum_{i \in S: e \in P_i} d_i \leq u_e$ for each edge e . In particular, Ls-Highway is the special case of NuLs-Highway where $d_i = 1$ for all i . Essentially the same reduction to UFP as mentioned above provides a $O(\log m)$ approximation also for NuLs-Highway.

1.1 Our Results and Technique

The main result of this paper is a PTAS for Ls-Highway (see Section 2).

► **Theorem 1.** *There is a deterministic PTAS for Ls-Highway.*

Our starting point is a hierarchical decomposition of G into subpaths (called *intervals*) of different levels as introduced in [25]. The whole path G forms the (only) interval of level 0. Then G is subdivided into $\Gamma = O_\epsilon(1)$ subintervals of level 1 such that for each subinterval the sum of the prices of the edges in the optimal solution is identical. Note that this decomposition depends on the unknown optimal solution and cannot be inferred directly from the input. Recursively, each interval of level ℓ is subdivided into Γ subintervals of level $\ell + 1$ with the latter property. A driver is said to be in level ℓ if its path is contained in an interval of level ℓ but not in an interval of level $\ell + 1$. The PTAS in [25] guesses this decomposition recursively. First, it guesses the partition of G into intervals of level 1. This implies which drivers are of level 0 and – using some additional arguments – for which of them the total price of the edges of their respective paths exceeds the budget. In more detail, using some shifting arguments one ensures that essentially each driver of level 0 crosses at least $1/\epsilon$ intervals of level 1 completely (and at most two such intervals partially). Since all intervals of level 1 have the same total price in the optimal solution, up to a factor of $1 + \epsilon$ this implies the amount that each driver of level 0 would have to pay if it is contained the optimal set of drivers. Then *all* drivers of level 0 are selected whose budget is not exceeded. Afterwards, the

algorithm continues recursively in the intervals of level 1. Importantly, in order to process an interval of a level ℓ , one does not need to know which drivers of smaller levels were selected previously. Instead, each arising subproblem can be described by an interval and a level. Therefore, the number of possible subproblems is bounded by a polynomial and the whole algorithm can easily be embedded into a polynomial time dynamic program.

In Ls-Highway this situation is drastically different. When we want to process an interval of level ℓ then it is not clear that we want to select all drivers of level ℓ whose budget is not exceeded since we might want to use the available edge capacity for drivers of larger levels instead. Also, we need to know the previously selected drivers of smaller levels since they might use capacity on the edges that we then cannot use for drivers of level ℓ anymore. Unfortunately, there is an exponential number of possibilities for which drivers have been selected before and hence we would get a super-polynomial number of possible subproblems. One could use the profiling technique in [5] in order to ensure that there are only a polynomial number of possibilities for the capacity taken by drivers from *each* previous level (with a small loss in the profit). However, since the number of levels is $\Omega(\log n)$ this yields a quasi-polynomial number of combinations for the used capacity from all levels together which is still too much.

At this point our main idea comes into play. We would like that the path of each driver of each level ℓ' starts and ends at the boundary vertex of an interval of level $\ell' + 1$. Then, when we process an interval of level ℓ it would be easy to describe the total capacity taken away from drivers of smaller levels: the total number of such drivers would suffice, knowing that each of them spans the entire interval. Therefore, consider the drivers of level ℓ that start or end in the middle of an interval G' of level $\ell + 1$, let us say there are m' such drivers. For each of them we try to delete a minimal set of drivers of level $\ell + 1$ or larger such that each edge of G' is used by at least one deleted driver. If we succeed then this frees up one unit of capacity along each edge of G' for each considered driver of level ℓ , and we can use this extra space to *forget* the actual portion of G' that is spanned by this driver. In other terms, we can imagine that its path spans the entire subinterval G' . If we do not succeed to delete enough drivers using some edge e then we allocate all remaining capacity on e to the drivers of level ℓ . In other words, the remaining capacity on each edge of G' is reduced *by* m' or *to zero*. Hence, when we process an interval G' of level $\ell + 1$ in our recursion then one number in $\{0, \dots, m\}$ suffices to describe by how much the capacity on each edge in G' is reduced due to drivers from smaller levels. We perform this deletion procedure for each level and each interval. This allows us then to devise a polynomial time dynamic program that computes a solution whose profit is at least as large as the profit of the remaining drivers.

To bound the cost of the above deletion step, by losing a factor $1 + \epsilon$ in the approximation ratio the construction in [25] ensures that the path of each driver i of a level ℓ spans at least $\Omega(1/\epsilon)$ intervals of level $\ell + 1$ and hence its profit is by a factor $\Omega(1/\epsilon)$ larger than the sum of the edge prices in an interval of level $\ell + 1$. Up to constant factors, the latter is the total profit of the drivers of level at least $\ell + 1$ that we delete for i in the procedure above. Therefore, the total profit due to the deleted tasks can be charged to i , losing only a factor of $1 + O(\epsilon)$.

The non-uniform demand case

Given the above PTAS, it is natural to address the non-uniform version of the problem. In particular:

► **Question 2.** *Is there a PTAS for NuLs-Highway?*

Using the hierarchical decomposition from above and ideas from [5] it is not hard to derive a QPTAS for the problem, i.e., a $(1 + \epsilon)$ -approximation that runs in quasi-polynomial time (at least for quasi-polynomially bounded capacities). Let $U_{max} = \max_{e \in G} \{u_e\}$ be the largest capacity.

► **Theorem 3.** *For any constant $\epsilon > 0$, there is a deterministic algorithm that computes a $(1 + \epsilon)$ -approximation for NuLs-Highway in time $(n \log U_{max})^{O_\epsilon(\log U_{max} \log m)}$.*

Also, using a folklore reduction to UFP one can obtain a $O(\log m)$ -approximation for NuLs-Highway.

► **Lemma 4.** *There is a polynomial-time deterministic $O(\log m)$ -approximation for NuLs-Highway.*

We were able to design a PTAS for the interesting special case of uniform budgets (see Section 3). Suppose that each driver has a budget of B . We partition G into blocks of total price B/ϵ each and ensure via a shifting argument that the path of essentially each driver is contained in some block. We guess this partition via a dynamic program step by step. For each block, on a high level we show that there is a near-optimal solution in which only $O_\epsilon(1)$ edges within the block have a non-zero price and hence we can guess these edges and their prices in polynomial time. The problem of selecting the drivers yields an instance of UFP in which each task uses one of the latter $O_\epsilon(1)$ edges. We invoke the known PTAS [23] for this case and obtain a $(1 + \epsilon)$ -approximation overall.

► **Theorem 5.** *There is a deterministic PTAS for NuLs-Highway in the special case that the budgets of all drivers are identical.*

1.2 Other Related Work

The *tollbooth* problem is the generalization of the highway problem where G is a tree. A $O(\log n)$ approximation was developed in [20], and later improved to $O(\log n / \log \log n)$ in [22]. Cygan et al. [18] present a $O(\log \log n)$ approximation for the case of uniform budgets. The tollbooth problem is APX-hard [26].

The highway and tollbooth problems belong to the family of pricing problems with single-minded customers and unlimited supply. Here we are given a set of customers: Each customer wants to buy a subset of items (*bundle*), if its total price does not exceed her budget. In the highway terminology, each driver is a subset of edges (rather than a path). For this problem a $O(\log n + \log m)$ approximation is given in [26]. This bound was refined in [9] to $O(\log L + \log B)$, where L denotes the maximum number of items in a bundle and B the maximum number of bundles containing a given item. Chalermsook et al. [12] showed that this problem is hard to approximate within $\log^{1-\epsilon} n$ for any constant $\epsilon > 0$. A $O(L)$ approximation is given in [3]. The latter approximation factor is asymptotically the best possible for constant values of L unless $P = NP$ as recently proved by Chalermsook et al. [13].

Elbassioni et al. [19] studied the limited-supply highway and tollbooth problems, however for *non-single-minded* drivers. Limited-supply pricing problems have also been studied in their *envy-free* version [26]: the goal here is to compute a maximum-profit pricing so that each client that can afford her bundle actually gets it. Cheung and Swamy [17] provided a $O(\log U)$ approximation for envy-free limited-supply highway with uniform capacities U . Observe that our algorithm does not guarantee envy-freeness. We also remark that requiring envy-freeness can substantially decrease the optimal profit, hence studying limited-supply pricing problems without this additional constraint makes sense in many applications.

The NuLs-Highway problem has several aspects in common with the well-studied *Unsplittable Flow on a Path* problem (UFP). In this problem we are given a path graph $G = (V, E)$, with edge capacities $\{u_e\}_{e \in E}$, and a set of tasks T , where each task i is characterized by a demand d_i , a subpath P_i of G , and a weight w_i . The goal is to select a maximum weight subset S of tasks such that the total demand $\sum_{i \in S: e \in P_i} d_i$ of selected tasks using each edge e is at most u_e . The current best approximation for this problem is $5/3 + \epsilon$ [24], improving on earlier results [2, 10, 6, 27, 15, 11, 8, 4]. The problem also admits a QPTAS [5, 7]. There is also a line of research on finding LP relaxations with small integrality gap for UFP [1, 11, 14].

1.3 Preliminaries

For any positive integer q let $[q] := \{1, 2, \dots, q\}$. We are given an $\epsilon > 0$ and assume w.l.o.g. that $1/(2\epsilon)$ is integral and $\epsilon \leq 1/2$. Let (OPT, p^*) denote an optimum solution to the considered instance, with drivers OPT and prices p^* , and opt be its profit. W.l.o.g., OPT contains only drivers with strictly positive profit. Standard reductions (see e.g. [25]) imply the following.

► **Lemma 6.** *By losing a factor $1 + \epsilon$ in the approximation, we can reduce in polynomial time a given instance of Ls-Highway to an instance of the same problem with $O(m^2/\epsilon)$ edges such that: (1) Budgets are integers between 1 and $\frac{m}{\epsilon}$; (2) Optimal prices take values in $\{0, 1\}$.*

Given the above reduction, we can assume that the sum P^* of the optimal prices is known by trying all the $O(m^2/\epsilon)$ possibilities.

For each edge e let $D_e := \{i \in D : e \in P_i\}$ be the drivers whose path contains e , and, for a subpath G' , $D(G') := \{i \in D : P_i \subseteq G'\}$ be the drivers whose path is contained in G' . Given prices p and a subpath G' , we let $p(G') = \sum_{e \in G'} p_e$. Given a driver i and prices p , we let the associated profit $\text{pro}(i, p)$ be $p(P_i)$ if this quantity is at most B_i , and 0 otherwise. For a subset of drivers S , $\text{pro}(S, p) = \sum_{i \in S} \text{pro}(i, p)$ is the total profit of those drivers. In case of non-uniform demands, we define $d(S') := \sum_{i \in S'} d_i$.

2 A PTAS for Ls-Highway

In this section we present our PTAS for Ls-Highway.

2.1 Hierarchical decomposition

Consider the input instance after applying the preprocessing step from Lemma 6, with optimal solution (OPT, p^*) . We next describe how to extract an almost optimal solution $\text{OPT}' \subseteq \text{OPT}$ with a convenient structure. Here we use the same construction as in [25].

Let $\Gamma = (1/\epsilon)^{1/\epsilon}$ and $\gamma = 1/(2\epsilon)$. We add dummy edges on the right of G (w.l.o.g. having a price of 1 each in the optimal solution) such that we can assume that $P^* = \Gamma^{\ell^*}$ for some integer $\ell^* = O_\epsilon(\log m)$. Since $n \leq \frac{m^2}{\epsilon}$ we can guess in polynomial time the number of dummy edges that we need and the resulting value of P^* . Let $x \in \{1, \dots, P^*\}$ and $y \in \{1, \dots, 1/\epsilon\}$ be two parameters to be fixed later. We append $P^* \cdot ((1/\epsilon)^y - 1) - x$ additional dummy edges to the left of G and x additional dummy edges to the right of G , resp., and we assume w.l.o.g. that p^* assigns a price of 1 to each one of them. To simplify the notation, we denote by G the resulting path, by p^* the resulting pricing, and by P^* the sum of prices in p^* . Observe that now $P^* = \Gamma^{\ell^*} (1/\epsilon)^y$.

Based on p^* , we define a hierarchical decomposition of G into nested subpaths (*inervals*). The starting point is the interval G of level 0. Given an interval G' of level ℓ , we partition it into Γ subintervals G'_1, \dots, G'_Γ of level $\ell + 1$, with uniform price. Observe that intervals of level ℓ have price $P^\ell := P^*/\Gamma^\ell = \Gamma^{\ell^* - \ell}(1/\epsilon)^y$. We stop the recursion at intervals of level ℓ^* , which have constant price $(1/\epsilon)^y = O_\epsilon(1)$.

For each interval G' we denote by $\ell(G')$ its level. We say that a driver i is at level ℓ if P_i is fully contained in an interval of level ℓ but in no interval of level $\ell + 1$. For each driver i , we let $\ell(i)$ be its level and $q(i)$ be the number of intervals of level $\ell(i) + 1$ which are fully contained in P_i . Based on the above decomposition and notation, we define an approximate profit function pro^* for each driver i of level $\ell(i) < \ell^*$ as follows

$$\text{pro}^*(i) = \begin{cases} 0 & \text{if } q(i) < \gamma \text{ or } q(i) \cdot P^{\ell(i)+1} > B_i \\ q(i) \cdot P^{\ell(i)+1} & \text{otherwise.} \end{cases} \quad (1)$$

For drivers i of level ℓ^* , we use the standard definition of profit, i.e. $\text{pro}^*(i) = p^*(P_i)$ for $p^*(P_i) \leq B_i$, and $\text{pro}^*(i) = 0$ otherwise. Intuitively, pro^* counts the profit of a driver i in level ℓ only if P_i spans many subintervals of level $\ell + 1$, i.e., at least γ many. For counting the profit, we ignore the two subintervals that P_i only partially overlaps with. Since P_i gets the full profit of at least γ subintervals, the difference is only a factor of $1 + O(\epsilon)$. On the other hand, it could be that $\text{pro}^*(i) > 0$ but i 's budget is exceeded. In this case it is still true that $\text{pro}(i, p^*) \geq \text{pro}^*(i)$ if i had a budget of $\frac{\gamma+2}{\gamma}B_i \leq (1 + O(\epsilon))B_i$. Therefore, intuitively we will pretend in the sequel that all drivers have a (larger) budget of $\frac{\gamma+2}{\gamma}B_i$ and repair this by scaling all edge prices at the very end. As usual, for $S \subseteq D$, $\text{pro}^*(S) = \sum_{i \in S} \text{pro}^*(i)$. We next let $\text{OPT}' \subseteq \text{OPT}$ be the drivers $i \in \text{OPT}$ with strictly positive $\text{pro}^*(i)$ (hence of profit at least $\gamma P^{\ell(i)+1}$).

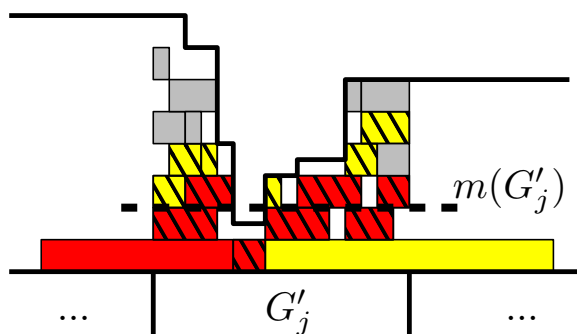
► **Lemma 7** ([25]). *There exist values of x and y such that $\text{pro}^*(\text{OPT}) = \text{pro}^*(\text{OPT}') \geq (1 - O(\epsilon))\text{opt}$.*

In the following we assume that the input graph is preprocessed according to the pair (x, y) given by Lemma 7: this is w.l.o.g. since we can try all the constantly many options. By pro^* we will denote the approximate profit function given by this choice.

2.2 A Structured Solution

At this point we introduce the most critical and novel idea in our PTAS. We extract from OPT' a large profit subset OPT'' that is even more structured. Intuitively, our goal is to limit the interaction between drivers of different levels. More formally, in OPT'' for each interval G' of some level ℓ' there exists a value m' such that on each edge e of G' the drivers of level ℓ' or larger use at most $\max\{0, u_e - m'\}$ units of capacity and the drivers of levels $\ell' - 1$ or smaller use the remaining capacity. Our algorithm will later select the drivers in the order of their levels, from small to large. Hence, in order to describe the capacity available on G' for the drivers level ℓ' or larger it suffices to know m' for which there are only $m + 1$ possibilities. This will be useful to define our algorithm as a polynomial time dynamic program.

Initially we set $\text{OPT}'' = \text{OPT}'$. Then we gradually move some drivers from OPT'' to a set of *deleted* drivers DEL. We will guarantee that the profit of deleted drivers is a small fraction of the profit of drivers that are still in OPT'' at the end of the process.



■ **Figure 1** For the red driver we delete a set of short drivers whose paths are contained in G'_j , depicted in striped red, that together completely cover G'_j . For the yellow driver we do not find such a set among the remaining drivers and delete drivers (depicted in striped yellow) that cover a maximal set of edges in G'_j . The gray drivers remain in the solution.

It is convenient to describe the construction of DEL in terms of a recursive procedure *delete*. This procedure, described in Algorithm 1, takes as input a tuple (G', ℓ', m') where G' is a subpath, $\ell' \in \{0, \dots, \ell^*\}$ is a level, and $m' \in \{0, \dots, m\}$ is some capacity. Note that w.l.o.g. we can assume that $u_e \leq m$ for each edge e . Furthermore, OPT'' and DEL are considered as global variables. We initialize $(\text{OPT}'', \text{DEL})$ to (OPT', \emptyset) , and run $\text{delete}(G, 0, 0)$.

The high-level idea behind *delete* is as follows. Intuitively, G' is some interval of level ℓ' , and m' is some uniform capacity that is *reserved* along G' to allocate drivers from previous levels whose path overlaps with G' . We remark that m' might exceed the capacity u_e available on some edge $e \in G'$, in which case drivers from level ℓ' or larger cannot use edge e (in other words, the *residual capacity* on edge e is $\max\{0, u_e - m'\}$). Consider the subdivision of G' into subintervals G'_1, \dots, G'_Γ . Let us focus on a specific G'_j , and consider the drivers of level ℓ' in $\text{OPT}'' \cap D(G')$ whose path intersects G'_j , let us denote them by $\text{OPT}''_{\ell'}(G'_j)$. Let $\text{OPT}''_{\ell', \text{part}}(G'_j)$ and $\text{OPT}''_{\ell', \text{span}}(G'_j)$ be the subset of them with $G'_j \not\subseteq P_i$ and $G'_j \subseteq P_i$, resp. In order to define the residual capacity for drivers in $D(G'_j)$ the drivers in $\text{OPT}''_{\ell', \text{span}}(G'_j)$ are not problematic: they use a uniform amount of capacity along G'_j . In order to handle the problematic drivers $\text{OPT}''_{\ell', \text{part}}(G'_j)$, the procedure *delete* removes some drivers from $\text{OPT}'' \cap D(G'_j)$ of level $\ell' + 1$ or larger. This leaves some free capacity that can be used to *ignore* the exact extent by which each $i \in \text{OPT}''_{\ell', \text{part}}(G'_j)$ overlaps with G'_j . Ideally, for each $i \in \text{OPT}''_{\ell', \text{part}}(G'_j)$, we would like to find a minimal set of drivers $\text{DEL}_i(G'_j) \subseteq \text{OPT}'' \cap D(G'_j)$ that spans G'_j , i.e., such that each edge of G'_j is used by at least one driver in $\text{DEL}_i(G'_j)$. However, there might not be enough drivers available for this in which case we rather take one such set with the largest possible span of edges of G'_j . This process is illustrated in Figure 1. After deleting all drivers in the sets $\text{DEL}_i(G'_j)$ for all $i \in \text{OPT}''_{\ell', \text{part}}(G'_j)$ we can safely set the (residual) capacity on edge e for drivers of level larger than ℓ' (whose path is contained in G'_j) to $\max\{0, u_e - m' - m_{\ell'}(G'_j)\}$ where $m_{\ell'}(G'_j) := |\text{OPT}''_{\ell'}(G'_j)|$. We then recurse in each subinterval G'_j of G' by calling $\text{delete}(G'_j, \ell' + 1, m' + m_{\ell'}(G'_j))$. We stop the recursion once we reach an interval of level ℓ^* in which case we do not delete any further drivers.

Consider OPT'' at the end of the root call $\text{delete}(G, 0, 0)$. This is obviously a feasible solution (being a subset of OPT'). Let us show that it has large profit.

► **Lemma 8.** *We have that $\text{pro}^*(\text{OPT}'') \geq (1 - O(\epsilon))\text{pro}^*(\text{OPT}')$*

Proof. Let us show that $\text{pro}^*(\text{DEL}) \leq \frac{4}{\gamma}\text{pro}^*(\text{OPT}'') = O(\epsilon)\text{pro}^*(\text{OPT}'')$. We use a charging argument. Consider an interval G' of level ℓ' and one of its subintervals G'_j . Note that, by construction, the total price over G' and G'_j is $P^{\ell'}$ and $P^{\ell'+1} = P^{\ell'}/\Gamma$, respectively. Consider

■ **Algorithm 1** Procedure to build the sets OPT'' and DEL .

```

delete( $G', \ell', m'$ )
1: if  $\ell' = \ell^*$  then
2:   halt;
3: Let  $G'_1, \dots, G'_\Gamma$  be the partition of  $G'$  into subintervals of level  $\ell + 1$ ;
4: for  $j = 1, \dots, \Gamma$  do
5:   Let  $\text{OPT}''_{\ell'}(G'_j) := \{i \in \text{OPT}'' \cap D(G') : \ell(i) = \ell', E(P_i) \cap E(G'_j) \neq \emptyset\}$ ;
6:   Let  $m_{\ell'}(G'_j) = |\text{OPT}''_{\ell'}(G'_j)|$ ;
7:   Let  $\text{OPT}''_{\ell', \text{part}}(G'_j) := \{i \in \text{OPT}''_{\ell'}(G'_j) : G'_j \not\subseteq P_i\}$ ;
8:   for every  $i \in \text{OPT}''_{\ell', \text{part}}(G'_j)$  do
9:     Let  $E_i(G'_j)$  be the edges used by  $\text{OPT}'' \cap D(G'_j)$ ;
10:    Let  $\text{DEL}_i(G'_j)$  be a minimal subset of  $\text{OPT}'' \cap D(G'_j)$  that spans  $E_i(G'_j)$ ;
11:    Set  $\text{OPT}'' \leftarrow \text{OPT}'' \setminus \text{DEL}_i(G'_j)$  and  $\text{DEL} \leftarrow \text{DEL} \cup \text{DEL}_i(G'_j)$ ;
12:   delete( $G'_j, \ell' + 1, m' + m_{\ell'}(G'_j)$ );

```

any $i \in \text{OPT}''_{\ell', \text{part}}(G'_j)$. Observe that i cannot be deleted in the next recursive calls, hence it finally belongs to OPT'' . Let us charge the loss due to the removal of $\text{DEL}_i(G'_j)$ to i .

By the minimality of $\text{DEL}_i(G'_j)$, each edge $e \in G'_j$ can be used by at most two drivers in $\text{DEL}_i(G'_j)$. It thus follows that $\text{pro}^*(\text{DEL}_i(G'_j)) \leq 2p^*(G'_j) \leq 2P^{\ell'+1}$. On the other hand, $\text{pro}^*(i) \geq \gamma P^{\ell'+1}$, hence $\text{pro}^*(\text{DEL}_i(G'_j)) \leq \frac{2}{\gamma} \text{pro}^*(i)$. Observe that each driver i in OPT'' of level ℓ' can be charged by at most two sets $\text{DEL}_i(G'_a)$ and $\text{DEL}_i(G'_b)$, associated with the (at most) two subintervals G'_a and G'_b of level $\ell' + 1$ that partially overlap with P_i (since the subintervals that are fully spanned by P_i do not charge i). It follows that

$$\text{pro}^*(\text{DEL}) = \sum_{G'_j, i} \text{pro}^*(\text{DEL}_i(G'_j)) \leq \frac{2}{\gamma} \sum_{G'_j, \ell', i \in \text{OPT}''_{\ell', \text{part}}(G'_j)} \text{pro}^*(i) \leq \frac{4}{\gamma} \text{pro}^*(\text{OPT}''). \quad \blacktriangleleft$$

One can show that, if in the recursion above a call $\text{delete}(G', \ell', m')$ arises, then in OPT'' on each edge e of G' the drivers of level ℓ' or larger use at most $\max\{0, u_e - m'\}$ units of capacity. We will use this property in our dynamic program below.

2.3 Dynamic program

We describe an algorithm that computes a solution with a profit of at least $\text{pro}^*(\text{OPT}'')$, pretending that each driver i has an increased budget of $\frac{\gamma+2}{\gamma} B_i$. Afterwards, we scale down the prices by a factor $\frac{\gamma}{\gamma+2}$ in order to respect the original budgets. Together with Lemma 8 this yields an approximation factor of $1 + O(\epsilon)$. For the sake of simplicity, in the sequel we will compute only the value of the desired solution while a straightforward extension yields an algorithm that finds the corresponding set of drivers and also the pricing for the edges.

A natural idea is to define a recursive algorithm that *guesses* the hierarchical decomposition into intervals and the values $m(G'_j)$ corresponding to OPT'' . Suppose we are given a tuple (G', ℓ', m') consisting of an interval G' , a level ℓ' , and an integer m' . The reader may imagine that in the hierarchical decomposition above G' is of level ℓ' and m' units of capacity are taken away on each edge of G' due to drivers of levels smaller than ℓ' . If $\ell' < \ell^*$ we guess the corresponding subdivision into subintervals G'_1, \dots, G'_Γ of level $\ell' + 1$ (each of them having length at least $P^{\ell'+1}$), and the associated values $m(G'_j)$, i.e., we try all possibilities for them. Via a reduction to UFP we select the drivers of level ℓ' : for each driver i with $P_i \subseteq G'$ but

$P_i \not\subseteq G'_j$ for each G'_j , we introduce a task i' with path $P_{i'} := P_i$ and demand $d_{i'} := 1$. For $q(i)$ being the number of intervals G'_j with $G'_j \subseteq P_i$ we define

$$w_{i'} := \begin{cases} 0 & \text{if } q(i) < \gamma \text{ or } q(i) \cdot P^{\ell'+1} > B_i \\ q(i) \cdot P^{\ell'+1} & \text{otherwise.} \end{cases} \quad (2)$$

Observe that to guarantee that we get a profit of $w_{i'}$ from driver i we would need that i has a budget of at least $\frac{\gamma+2}{\gamma} B_i$. This can be fixed at the end by scaling down prices by a factor $\frac{\gamma+2}{\gamma}$ (with a small profit loss). We define the edge capacities by $u'_e := \min\{m(G'_j), \max\{0, u_e - m'\}\}$ for each edge e in a subinterval G'_j . Since all drivers have unit demand this instance of UFP can be solved exactly in polynomial time (see, e.g., [16]). Then we recurse on each interval G'_j such that the corresponding subproblem consists of the tuple $(G'_j, \ell' + 1, m' + m(G'_j))$. Finally, the solution for (G', ℓ', m') is the most profitable solution obtained in this way over all of the guesses above.

If we are given a tuple (G', ℓ', m') with $\ell' = \ell^*$ (the reader may again imagine that G' is an interval of level ℓ^*) then we guess directly the optimal pricing p^* which is one of the polynomially many options to assign a total price of $(1/\epsilon)^y = O_\epsilon(1)$ to the edges of G' such that each edge gets a price in $\{0, 1\}$. Selecting the drivers yields again an instance of UFP. For each driver i with $P_i \subseteq G'$ we introduce a task i' with path $P_{i'} := P_i$, demand $d_{i'} := 1$, and weight $w_{i'} = p(B_i)$ if $p(B_i) \leq B_i$ and $w_{i'} = 0$ otherwise. Each edge e has a capacity of $u'_e := \max\{0, u_e - m'\}$. Again, since all drivers have unit demand we can solve this instance of UFP in polynomial time [16]. The solution for (G', ℓ', m') is then the most profitable solution over all guesses. We return the solution to $(G, 0, 0)$.

As it is described above, this algorithm does not have polynomial running time since in each subproblem we enumerate a polynomial number of guesses and the recursion depth is $\Omega(\log n)$. However, each recursive call is specified by a tuple (G', ℓ', m') and there are only a polynomial number of those. Hence, we can embed our algorithm into a polynomial time dynamic program, see Algorithm 2. For each cell (G', ℓ', m') denote by $DP(G', \ell', m')$ the value stored in it.

■ **Algorithm 2** Dynamic program to approximate Ls-Highway. Here G' denote a subpath of G of length at least $P^{\ell'}$, $\ell' \in \{0, \dots, \ell^*\}$ a level, and $m' \in \{0, \dots, m\}$ a capacity.

compute $DP(G', \ell', m')$

- 1: **if** $\ell' < \ell^*$ **then**
- 2: **for** all possible subdivisions of G' into subpaths G'_1, \dots, G'_Γ of length at least $P^{\ell'+1}$ **each do**
- 3: **for** all possible values $m(G'_j) \in \{0, \dots, m\}$, $j = 1, \dots, \Gamma$ **do**
- 4: construct the UFP instance I' associated with $(G', m', \{G'_j\}_j, \{m(G'_j)\}_j)$;
- 5: solve I' optimally, let $wufp(I')$ be the resulting profit
- 6: compute

$$w(G', m', \{G'_j\}_j, \{m(G'_j)\}_j) := wufp(I') + \sum_{j=1}^{\Gamma} DP(G'_j, \ell' + 1, m' + m(G'_j))$$

- 7: $DP(G', \ell', m') \leftarrow$ largest value $w(G', m', \{G'_j\}_j, \{m(G'_j)\}_j)$ computed in Step 6
 - 8: **if** $\ell' = \ell^*$ **then**
 - 9: **for** all possible assignments $p = \{0, 1\}^{E(G')}$ with $p(G') = P^{\ell^*} = (1/\epsilon)^y$ **do**
 - 10: construct the UFP instance I' associated with (G', m', p) ;
 - 11: solve I' optimally, let $wufp(I')$ be the resulting profit
 - 12: $DP(G', \ell', m') \leftarrow$ largest value $wufp(I')$ computed in Step 11
-

54:10 Packing Cars into Narrow Roads: PTASs for Limited Supply Highway

Let us consider the recursive partition of G that corresponds to $(G, 0, 0)$, i.e., the intervals of the partition achieving the maximum in Line 7 and recursively their subpartitions achieving the maximum in their respective subproblems. Let \mathcal{G} be the intervals in this partition. For a given $G' \in \mathcal{G}$ we let $\ell'(G')$ and $m'(G')$ denote the associated values of ℓ' and m' . Furthermore, if $\ell'(G') < \ell^*$, let $\{G'_j\}_j$ and $\{m(G'_j)\}_j$ be the corresponding values achieving the maximum in Step 7. By $\text{ALG}(G')$ we denote the UFP solution corresponding to the cell $(G', \ell'(G'), m'(G'))$ that achieves the maximum value in Step 7 or 12. Note that the solution associated with $(G, 0, 0)$ is $\text{ALG} = \cup_{G' \in \mathcal{G}} \text{ALG}(G')$. By p^{ALG} we denote the pricing induced by the values p achieving the maximum in Step 12.

► **Lemma 9.** *ALG respects the capacity constraints.*

Proof. For a given $G' \in \mathcal{G}$, let $\text{ALG}^\downarrow(G') := \cup_{G'' \in \mathcal{G}: G'' \subseteq G'} \text{ALG}(G'')$ be the union of all the UFP solutions corresponding to subintervals contained in G' (G' included). We will show by induction on decreasing values of $\ell'(G')$ that $\text{ALG}^\downarrow(G')$ is a feasible solution w.r.t. residual capacities $\max\{0, u_e - m'(G')\}$ i.e.

$$|D_e \cap \text{ALG}^\downarrow(G')| \leq \max\{0, u_e - m'(G')\}, \quad \forall e \in G'.$$

The claim then follows since $m'(G) = 0$ and $\text{ALG}^\downarrow(G) = \text{ALG}$.

For the base case $\ell'(G') = \ell^*$ this is true by the definition of the edges capacities for the case that $\ell' = \ell^*$. Suppose next the claim is true up to the value $\ell' + 1$, and consider G' with $\ell'(G') = \ell'$. Consider any edge $e \in G'_j$, for some $j \in [\Gamma]$.

$$|D_e \cap \text{ALG}^\downarrow(G'_j)| \leq \max\{0, u_e - m'(G') - m(G'_j)\}.$$

By construction and the definition of the edge capacities for the case that $\ell' < \ell^*$ we have

$$|D_e \cap \text{ALG}(G'_j)| \leq \min\{m(G'_j), \max\{0, u_e - m'(G')\}\}.$$

Thus

$$\begin{aligned} |D_e \cap \text{ALG}^\downarrow(G')| &= |D_e \cap \text{ALG}(G')| + |D_e \cap \text{ALG}^\downarrow(G'_j)| \\ &\leq \min\{m(G'_j), \max\{0, u_e - m'(G')\}\} + \max\{0, u_e - m'(G') - m(G'_j)\} \\ &\leq \max\{0, u_e - m'(G')\}, \end{aligned}$$

where the last inequality follows easily by distinguishing the cases $u_e - m'(G') \leq 0$, $0 < u_e - m'(G') \leq m(G'_j)$, and $u_e - m'(G') > m(G'_j)$. ◀

The proof of the following lemma follows by constraining the choices of the algorithm in order to mimic the construction of OPT'' . The crucial step is to show that, for a subproblem (G', ℓ', m') (corresponding to an interval G' in the hierarchical decomposition due to Section 2.1) the drivers in $\text{OPT}'' \cap D(G')$ of level ℓ' (denote them by $\text{OPT}''(G', \ell')$) define a feasible solution for the associated UFP instance whose weight is precisely $\text{pro}^*(\text{OPT}''(G', \ell'))$ by the definition of the tasks weights of these instances.

► **Lemma 10.** $DP(G, 0, 0) = \text{pro}^*(\text{ALG}) \geq \text{pro}^*(\text{OPT}'')$.

Finally, we scale down the price on each edge by a factor $\frac{\gamma+2}{\gamma} \leq 1 + O(\epsilon)$, i.e. we return the solution $(\text{ALG}, \frac{\gamma}{\gamma+2} p^{\text{ALG}})$. This way, all drivers in ALG respect the original budgets and we achieve a profit almost as large as $DP(G, 0, 0)$.

► **Lemma 11.** $\text{pro}(\text{ALG}, \frac{\gamma}{\gamma+2} p^{\text{ALG}}) \geq \frac{\gamma}{\gamma+2} DP(G, 0, 0)$.

Proof. It is sufficient to show that, after scaling prices, the budget of each driver $i \in \text{ALG}$ is not exceeded. It then follows that the profit associated with i is precisely $\frac{\gamma}{\gamma+2}$ times its weight w_i in the corresponding UFP instance. W.l.o.g. we can assume that $w_i > 0$. If the level ℓ of i is ℓ^* , the claim is trivial since $\text{pro}(i, p^{\text{ALG}}) = w_i$. In other words, the budget of i is respected even without scaling the prices. Otherwise, with the usual notation, by definition we have that $q(i) \geq \gamma$ and $q(i) \cdot P^\ell \leq B_i$. By construction the total price associated with i is

$$p^{\text{ALG}}(P_i) \leq (q(i) + 2)P^\ell \leq \frac{\gamma + 2}{\gamma} q(i)P^\ell \leq \frac{\gamma + 2}{\gamma} B_i.$$

Hence $\frac{\gamma}{\gamma+2} p^{\text{ALG}}$ does not violate the budget of i as required. \blacktriangleleft

Our algorithm runs in polynomial time since we have a polynomial number of DP-cells and the computation for each takes polynomial time. Now the proof of Theorem 1 follows immediately from Lemmas 8-11.

3 A PTAS for NuLs-Highway with Uniform Budgets

In this section we present a PTAS for NuLs-Highway when all drivers have the same budget B . It is not hard (modulo technicalities) to extend our result to the case that the ratio of largest to smallest budget is upper bounded by a given constant.

We will use the following folklore result for the highway problem (and more generally for item pricing problems), that immediately extends to Ls-Highway and NuLs-Highway.

► **Lemma 12** (Close To Budget Lemma). *Given any $\alpha \in (0, 1]$, in any optimal solution (OPT, p^*) to NuLs-Highway at least a fraction $(1 - \alpha)$ of the profit is due to drivers whose profit is at least α times their budget.*

Proof. Assume by contradiction the claim is not true, and consider the drivers $i \in S \subseteq \text{OPT}$ whose profit in p^* is less than $\alpha \cdot B_i$. Then the pricing p^*/α achieves a profit larger than OPT from S , a contradiction. \blacktriangleleft

Let us first show that a solution with a convenient structure exists. Let (OPT, p^*) be an optimal solution. Using Lemma 6 we can assume that the price of each edge is in $\{0, 1\}$ and that $B \in \{1, \dots, m/\epsilon\}$. Let P^* be the sum of the optimal prices, and h^* be the smallest integer such that $P^* \leq (h^* - 1)\frac{B}{\epsilon}$. We guess P^* and hence we then also know h^* . For a choice of $x \in \{0, \dots, \frac{1}{\epsilon}B - 1\}$ to be defined later, we append x edges to the left of the input graph G , and $y = (h^* - 1)\frac{B}{\epsilon} - P^* + \frac{1}{\epsilon}B - x$ edges to its right. W.l.o.g. we assume that each new edge has a price of 1 in (OPT, p^*) . For simplicity, we still denote by G the resulting graph, by p^* its optimal pricing, by P^* the total price of all edges and we define $h^* := P^*\epsilon/B$. Observe that $p^*(G) = h^*\frac{B}{\epsilon}$.

By $\text{OPT}' \subseteq \text{OPT}$ we denote the drivers i whose profit in p^* is at least $\epsilon \cdot B_i$. By applying Lemma 12 with $\alpha = \epsilon$ one has that $\text{pro}(\text{OPT}', p^*) \geq (1 - \epsilon)\text{opt}$. We next define a solution $(\text{APX}, p^{\text{apx}})$, with $\text{APX} \subseteq \text{OPT}'$. Subdivide G in h^* subpaths B_j (blocks) with total price exactly $\frac{B}{\epsilon}$ each. Discard all drivers $i \in \text{OPT}'$ whose path P_i contains edges in two different blocks: let APX be the remaining drivers. Subdivide each B_j into $1/\epsilon^3$ subpaths $B_{j,k}$ of optimal price exactly $\epsilon^2 B$ each (sub-blocks). In p^{apx} set the price of the rightmost edge in each sub-block to $\frac{1}{1+\epsilon} \cdot \epsilon^2 B$ and the price of any other edge to zero (note that we use fractional prices even though we assumed the optimal solution to have prices in $\{0, 1\}$).

Let us show that the profit of the new solution is large enough for a proper choice of x .

► **Lemma 13.** *There is a choice of x in the above construction such that $\text{pro}(APX, p^{apx}) \geq (1 - O(\epsilon)) \text{opt}$.*

We devise now a dynamic program that computes a solution with a profit of at least $\text{pro}(APX, p^{apx})$. Intuitively, it guesses step by step the above partition into blocks. Then for each block B it guesses its partition into subblocks, sets a price of $\frac{1}{1+\epsilon} \cdot \epsilon^2 B$ to the rightmost edge of each subblock and computes a subset of drivers from $D(B)$ maximizing the profit from the selected drivers. The problem of selecting these drivers yields special instances of UFP (one for each block) in which there are $1/\epsilon^3$ special edges (the edges with non-zero price) such that each input task uses at least one of them (all other drivers yield zero profit and can be discarded). We invoke the known PTAS for this special case [23].

Formally, first we guess the value for $x \in \{0, \dots, \frac{1}{\epsilon} B - 1\}$ due to Lemma 12. Observe that $B \leq m/\epsilon$ due to Lemma 5 and hence there are only m/ϵ^2 options for x . We start by preprocessing the instance as described before for the considered x : let N be the final number of edges, and let us label them from 1 to N from left to right. Let $G_{\ell,r}$ be the subpath of G with leftmost edge ℓ and rightmost edge r . The DP table is indexed by pairs (r, h) where r is some edge and $h \in \{1, \dots, h^*\}$. Intuitively, the value of $DP(r, h)$ is the maximum profit that is achievable by drivers whose path is contained in $G_{1,r}$ in the following way:

1. We divide $G_{1,r}$ into h blocks B_j , subdivide each block into $1/\epsilon^3$ sub-blocks, and assign the price $\frac{\epsilon^2 B}{1+\epsilon}$ to the rightmost edge of each sub-block (and 0 otherwise).
2. We select a set of drivers such that the path of each driver is fully contained in some block.

As usual, we can associate to $DP(r, h)$ a specific solution of the same profit. At the end, we output the solution in the cell $DP(N, h^*)$.

Consider a given DP-cell $DP(r, h)$. For all values ℓ with $1 \leq \ell < r$ we do the following: we partition $G_{1,r}$ into a block $G_{\ell,r}$ and a remaining part $G_{1,\ell-1}$. We consider all the possible $O(n^{1/\epsilon^3})$ ways to subdivide $G_{\ell,r}$ into sub-blocks $B_{\ell,r}^1, \dots, B_{\ell,r}^{1/\epsilon^3}$ such that none of them is empty. For any such choice, we define a UFP instance $UFP(\{B_{\ell,r}^k\}_k)$ as follows. The graph is $G_{\ell,r}$, with the corresponding edge capacities u_e , $e \in G_{\ell,r}$. For each driver i with $P_i \subseteq G_{\ell,r}$, we define a task i' with path $P_{i'} := P_i$ and demand $d_{i'} := d_i$. We define its weight $w_{i'}$ as follows: assign price $\frac{\epsilon^2 B}{1+\epsilon}$ to the rightmost edge in each sub-block; set $w_{i'}$ to the total price on the edges of $P_{i'}$ if this is at most B_i , and 0 otherwise. We discard a task i' if with $w_{i'} = 0$.

Note that in this instance of UFP each task must use one of the $1/\epsilon^3$ edges with non-zero price. We invoke the PTAS in [23, Theorem 3.3] for this special case. Let $\text{alg}(\ell, r)$ be the maximum weight of any computed UFP solution for this choice of ℓ (over all partitions $B_{\ell,r}^1, \dots, B_{\ell,r}^{1/\epsilon^3}$). Observe that this value depends only on ℓ and r . If $r - \ell + 1 < 1/\epsilon^3$ then there can be no partition in which all sub-blocks are non-empty and therefore we set $\text{alg}(\ell, r) = 0$. Given the above quantities, we define

$$DP(r, h) := \max_{1 \leq \ell < r} \{\text{alg}(\ell, r) + DP(\ell - 1, h - 1)\}$$

where we define $DP(0, h) = 0$ for all h and $DP(r, 0) = 0$ for all r . Finally, we output the solution in the DP-cell $DP(N, h^*)$.

References

- 1 Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. Constant Integrality Gap LP Formulations of Unsplittable Flow on a Path. In *International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 25–36, 2013.
- 2 Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. A Mazing $2+\epsilon$ Approximation for Unsplittable Flow on a Path. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 26–41, 2014.

- 3 Maria-Florina Balcan and Avrim Blum. Approximation algorithms and online mechanisms for item pricing. In *ACM Conference on Electronic Commerce*, pages 29–35, 2006.
- 4 N. Bansal, Z. Friggstad, R. Khandekar, and R. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 702–709, 2009.
- 5 Nikhil Bansal, Amit Chakrabarti, Amir Epstein, and Baruch Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *ACM Symposium on Theory of computing (STOC)*, pages 721–729, 2006.
- 6 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.
- 7 Jatin Batra, Naveen Garg, Amit Kumar, Tobias Mömke, and Andreas Wiese. New Approximation Schemes for Unsplittable Flow on a Path. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 47–58, 2015.
- 8 P. Bonsma, J. Schulz, and A. Wiese. A Constant Factor Approximation Algorithm for Unsplittable Flow on Paths. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 47–56, 2011.
- 9 Patrick Briest and Piotr Krysta. Single-minded unlimited supply pricing on sparse instances. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1093–1102, 2006.
- 10 Gruia Calinescu, Amit Chakrabarti, Howard J. Karloff, and Yuval Rabani. Improved Approximation Algorithms for Resource Allocation. In *International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 401–414, 2002.
- 11 Amit Chakrabarti, Chandra Chekuri, Anupam Gupta, and Amit Kumar. Approximation Algorithms for the Unsplittable Flow Problem. *Algorithmica*, 47(1):53–78, 2007.
- 12 Parinya Chalermsook, Julia Chuzhoy, Sampath Kannan, and Sanjeev Khanna. Improved Hardness Results for Profit Maximization Pricing Problems with Unlimited Supply. In *International Workshop on Approximation, Randomization, and Combinatorial Optimization (APPROX-RANDOM)*, pages 73–84, 2012.
- 13 Parinya Chalermsook, Bundit Laekhanukit, and Danupon Nanongkai. Independent Set, Induced Matching, and Pricing: Connections and Tight (Subexponential Time) Approximation Hardnesses. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 370–379, 2013.
- 14 C. Chekuri, A. Ene, and N. Korula. Unsplittable Flow in Paths and Trees and Column-Restricted Packing Integer Programs. In *International Workshop on Approximation, Randomization, and Combinatorial Optimization (APPROX-RANDOM)*, pages 42–55, 2009.
- 15 Chandra Chekuri, Marcelo Mydlarz, and F. Bruce Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms*, 3(3), 2007.
- 16 Chandra Chekuri, Marcelo Mydlarz, and F. Bruce Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms (TALG)*, 3(3):27, 2007.
- 17 Maurice Cheung and Chaitanya Swamy. Approximation Algorithms for Single-minded Envy-free Profit-maximization Problems with Limited Supply. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 35–44, 2008.
- 18 Marek Cygan, Fabrizio Grandoni, Stefano Leonardi, Marcin Pilipczuk, and Piotr Sankowski. A Path-Decomposition Theorem with Applications to Pricing and Covering on Trees. In *European Symposium on Algorithms (ESA)*, pages 349–360, 2012.
- 19 Khaled M. Elbassioni, Mahmoud Fouz, and Chaitanya Swamy. Approximation Algorithms for Non-single-minded Profit-Maximization Problems with Limited Supply. In *Internet and Network Economics - 6th International Workshop, WINE 2010, Stanford, CA, USA, December 13-17, 2010. Proceedings*, pages 462–472, 2010.

- 20 Khaled M. Elbassioni, Rajiv Raman, Saurabh Ray, and René Sitters. On Profit-Maximizing Pricing for the Highway and Tollbooth Problems. In *International Symposium on Algorithmic Game Theory (SAGT)*, pages 275–286, 2009.
- 21 Khaled M. Elbassioni, René Sitters, and Yan Zhang. A Quasi-PTAS for Profit-Maximizing Pricing on Line Graphs. In *European Symposium on Algorithms (ESA)*, pages 451–462, 2007.
- 22 Iftah Gamzu and Danny Segev. A Sublogarithmic Approximation for Highway and Tollbooth Pricing. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 582–593, 2010.
- 23 Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. To Augment or Not to Augment: Solving Unsplittable Flow on a Path by Creating Slack. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2411–2422, 2017.
- 24 Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. A $(5/3 + \epsilon)$ -approximation for unsplittable flow on a path: placing small tasks into boxes. In *ACM Symposium on Theory of Computing (STOC)*, pages 607–619, 2018.
- 25 Fabrizio Grandoni and Thomas Rothvoß. Pricing on Paths: A PTAS for the Highway Problem. *SIAM J. Comput.*, 45(2):216–231, 2016.
- 26 Venkatesan Guruswami, Jason D. Hartline, Anna R. Karlin, David Kempe, Claire Kenyon, and Frank McSherry. On profit-maximizing envy-free pricing. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1164–1173, 2005.
- 27 Cynthia A. Phillips, R. N. Uma, and Joel Wein. Off-line admission control for general scheduling problems. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 879–888, 2000.

Engineering Negative Cycle Canceling for Wind Farm Cabling

Sascha Gritzbach 

Karlsruhe Institute of Technology, Germany
sascha.gritzbach@kit.edu

Torsten Ueckerdt 

Karlsruhe Institute of Technology, Germany
torsten.ueckerdt@kit.edu

Dorothea Wagner 

Karlsruhe Institute of Technology, Germany
dorothea.wagner@kit.edu

Franziska Wegner 

Karlsruhe Institute of Technology, Germany
franziska.wegner@kit.edu

Matthias Wolf 

Karlsruhe Institute of Technology, Germany
matthias.wolf@kit.edu

Abstract

In a wind farm turbines convert wind energy into electrical energy. The generation of each turbine is transmitted, possibly via other turbines, to a substation that is connected to the power grid. On every possible interconnection there can be at most one of various different cable types. Each cable type comes with a cost per unit length and with a capacity. Designing a cost-minimal cable layout for a wind farm to feed all turbine production into the power grid is called the WIND FARM CABLING PROBLEM (WCP).

We consider a formulation of WCP as a flow problem on a graph where the cost of a flow on an edge is modeled by a step function originating from the cable types. Recently, we presented a proof-of-concept for a negative cycle canceling-based algorithm for WCP [14]. We extend key steps of that heuristic and build a theoretical foundation that explains how this heuristic tackles the problems arising from the special structure of WCP.

A thorough experimental evaluation identifies the best setup of the algorithm and compares it to existing methods from the literature such as MIXED-INTEGER LINEAR PROGRAMMING (MILP) and Simulated Annealing (SA). The heuristic runs in a range of half a millisecond to under two minutes on instances with up to 500 turbines. It provides solutions of similar quality compared to both competitors with running times of one hour and one day. When comparing the solution quality after a running time of two seconds, our algorithm outperforms the MILP- and SA-approaches, which allows it to be applied in interactive wind farm planning.

2012 ACM Subject Classification Mathematics of computing → Network flows; Mathematics of computing → Graph algorithms; Mathematics of computing → Network optimization

Keywords and phrases Negative Cycle Canceling, Step Cost Function, Wind Farm Planning

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.55

Funding This work was funded (in part) by the Helmholtz Program Storage and Cross-linked Infrastructures, Topic 6 Superconductivity, Networks and System Integration, by the Helmholtz future topic Energy Systems Integration, and by the German Research Foundation (DFG) as part of the Research Training Group GRK 2153: Energy Status Data – Informatics Methods for its Collection, Analysis and Exploitation.

Acknowledgements We thank our colleagues Lukas Barth and Marcel Radermacher for valuable discussions and Lukas Barth for providing a code interface for simultaneous MILP-solver experiments.



© Sascha Gritzbach, Torsten Ueckerdt, Dorothea Wagner, Franziska Wegner, and Matthias Wolf; licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 55; pp. 55:1–55:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Wind energy becomes increasingly important to help reduce effects of climate change. As of 2017, 11.6% of the total electricity demand in the European Union is covered by wind power [24]. Across the Atlantic, the state of New York aims at installing 2.4 GW of offshore wind energy capacity by 2030, which could cover the demand of 1.2 million homes [19].

In an offshore wind farm a set of turbines generate electrical energy. From offshore substations the energy is transmitted via sea cables to an onshore grid point. One of the biggest wind farms currently planned is Hornsea Project Three in the North Sea with up to 300 turbines and twelve substations [1]. To transport turbine production to the substations, a system of cables links turbines to substations (*internal cabling*) where multiple turbines may be connected in series. The designer of a wind farm has various cable types available, each of which with respective costs and thermal capacities. The latter restricts the amount of energy that can be transmitted through a cable. Planning a wind farm as a whole consists of various steps, including determining the locations for turbines and substations, laying out the connections from substations to the grid point, and designing the internal cabling. The planning process comes with a high level of complexity, which automated approaches struggle with [23]. Therefore, one might opt for decoupling the planning steps. We call the task of finding a cost-minimal internal cabling of a wind farm with given turbine and substation positions, as well as given turbine production and substation capacities, the WIND FARM CABLING PROBLEM (WCP). Since WCP is a generalization of the \mathcal{NP} -hard problem CAPACITATED MINIMUM SPANNING TREE [21], it is \mathcal{NP} -hard as well.

Due to the overall cost of a wind farm, using one day of computation time or more arguably is a reasonable way to approach WCP. Such computation times, however, are not appropriate for an interactive planning process: Imagine a wind farm planner uses a planning tool which allows altering turbine positions to explore their influence on possible cable layouts. In that case, computation times of at most several seconds are desirable.

1.1 Contribution and Outline

We extend our recent proof-of-concept, in which negative cycle canceling is applied to a formulation of WCP as a network flow problem (cf. Section 3) with a step cost function representing the cable types [14]. The idea of negative cycle canceling is to iteratively identify cycles in a graph in which the edges are associated with the costs of (or gains from) changing the flow. Normally, a cycle of negative total cost corresponds to a way to decrease the cost of a previously found flow. Due to the step cost function, however, not every negative cycle helps improve a solution to WCP. We explore this and other issues for negative cycle canceling that arise from the step cost function in the flow problem formulation for WCP. We present a modification of the Bellman-Ford algorithm [3, 9] and build a theoretical foundation that explains how the modified algorithm addresses the aforementioned issues, e. g., by being able to identify cycles that actually improve a solution. This modification works on a subgraph of the line graph (cf. page 6) of the input graph and can be implemented in the same asymptotic running time as the original Bellman-Ford algorithm.

We further extend that heuristic by identifying two key abstraction layers and applying different strategies in those layers. Using different initializations is hinted at in the section on future work in [14]. We follow this hint and design eight concrete initialization strategies. In another layer, we propose a total of eight so-called “delta strategies” that specify the order in which different values for flow changes are considered.

In [14] we compared the Negative Cycle Canceling (NCC) algorithm to a MIXED-INTEGER LINEAR PROGRAM (MILP) using the MILP solver Gurobi with one-hour running times on benchmark sets from the literature [17]. We extend this evaluation by identifying the best of our variants and by comparing its results to the results of MILP experiments after running times of two seconds, one hour, and one day on the same benchmark sets. A running time of two seconds helps identify the usefulness of the NCC algorithm to an interactive planning process. The other running times stand for non-time-critical planning. We also compare the algorithm to an approach using Simulated Annealing [17] with different running times. The results show that our heuristic is very fast since it terminates on instances with up to 500 turbines in under two minutes. At two seconds our algorithm outperforms its competitors, making it feasible for interactive wind farm planning. Even with longer running times for the MILP- and SA-approaches, our algorithm yields solutions to WCP of similar quality but in tens of seconds.

In Section 2 we review existing work on WCP and negative cycle canceling. In Section 3 we define WCP as a flow problem. We give theoretical insights on the difference to standard flow problems and present and analyze our Negative Cycle Canceling algorithm in Section 4. An extensive experimental evaluation of the algorithm is given in Section 5. We conclude with a short summary of the results and outline possible research directions (see Section 6).

2 Related Work

In one of the first works on WCP, a hierarchical decomposition of the problem was introduced [4]. The layers relate to well-known graph problems and heuristics for various settings are proposed. Since then, considerable effort has been put into solving different variants of WCP. Exact solutions can be computed using MIXED-INTEGER LINEAR PROGRAM (MILP) formulations including various degrees of technical constraints, e. g., line losses, component failures, and wind stochasticity [18]. However, sizes of wind farms that are solved to optimality in reasonable time are small. Metaheuristics such as Genetic Algorithms [25, 6] or Simulated Annealing [17] can provide good but not necessarily optimal solutions in relatively short computation times.

We applied negative cycle canceling to a suitable flow formulation for WCP [14], but there is still an extensive agenda of open questions such as investigating the effect of other cable types, a comparison to existing heuristics, and using the solution as warm start for a MILP solver. Originally, negative cycle canceling is proposed in the context of minimum cost circulations when linear cost functions are considered [16]. The algorithm for the Minimum-Cost Flow Problem based on cycle canceling with strongly polynomial running time runs in $\mathcal{O}(nm(\log n) \min\{\log(nC), m \log n\})$ time on a network with n vertices, m edges, and maximum absolute value of costs C [12]. The bound for the running time of this algorithm was later tightened to $\Theta(\min\{nm \log(nC), nm^2\})$ [22]. Negative cycle canceling has also been used for problems with non-linear cost functions. Among these are multicommodity flow problems with certain non-linear yet convex cost functions based on a queueing model [20] and the Capacity Expansion Problem for multicommodity flow networks with certain non-convex and non-smooth cost functions [7]. A classic algorithm for finding negative cycles is the Bellman-Ford algorithm [3, 9] with heuristic improvements [13, 11]. An experimental evaluation of these heuristics and other negative cycle detection algorithms is given in [5].

A step cost function similar to the one in WCP appears in a multicommodity flow problem, for which exact solutions can be obtained by a procedure based on Benders Decomposition [10]. However, this procedure is only evaluated on instances with up to 20 vertices and 37 edges and

some running times exceed 13 hours. While our approach does not guarantee to solve WCP to optimality, our evaluation shows that the solution quality is very good compared to the MILP with running times not exceeding two minutes on wind farms with up to 500 turbines.

3 Model

The model presented in this paper is based on an existing flow model for WCP [14]. We briefly recall the model. Given a wind farm, let V_T and V_S be the sets of turbines and substations, respectively. We define a vertex set V of a graph by $V = V_T \cup V_S$. For any two vertices u and v that can be connected by a cable in the wind farm, we define exactly one directed edge $e = (u, v)$, where the direction is chosen arbitrarily. We obtain a directed graph $G = (V, E)$ with $V = V_T \cup V_S$ and $E \subseteq (V \times V) \setminus (V_S \times V_S)$ such that $(u, v) \in E$ implies $(v, u) \notin E$. There are no edges between any two substations since we consider the wind farm planning step in which all positions of turbines and substations, as well as the cabling from substations to the onshore grid point have been fixed. We assume that all turbines generate one unit of electricity. Note that our algorithm can be easily generalized to handle non-uniform integral generation. Substations have a capacity $\text{cap}_{\text{sub}}: V_S \rightarrow \mathbb{N}$ representing the maximum amount of turbine production they can handle and each edge has a length given by $\text{len}: E \rightarrow \mathbb{R}_{\geq 0}$ representing the geographic distance between the endpoints of the edge.

A *flow* on G is a function $f: E \rightarrow \mathbb{R}$ and for an edge (u, v) with $f(u, v) > 0$ (resp. < 0), we say that $f(u, v)$ units of flow go from u to v (resp. $-f(u, v)$ units go from v to u). For a flow f and a vertex u we define the *net flow in u* by $f_{\text{net}}(u) = \sum_{(w,u) \in E} f(w, u) - \sum_{(u,w) \in E} f(u, w)$. A flow f is *feasible* if the conditions on flow conservation for both turbines (Equation (1)) and substations (Equation (2)) are satisfied and if there is no outflow from any substation (Equations (3) and (4)).

$$f_{\text{net}}(u) = -1 \quad \forall u \in V_T, \quad (1)$$

$$f_{\text{net}}(v) \leq \text{cap}_{\text{sub}}(v) \quad \forall v \in V_S, \quad (2)$$

$$f(u, v) \geq 0 \quad \forall (u, v) \in E : v \in V_S, \quad (3)$$

$$f(v, u) \leq 0 \quad \forall (v, u) \in E : v \in V_S. \quad (4)$$

Let $c: \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$ be a non-decreasing, left-continuous step function with $c(0) = 0$. This function represents the cable costs and $\sup\{x \in \mathbb{R}_{\geq 0} : c(x) < \infty\}$ is the maximum cable capacity, which we assume to be a natural number. Note that such a function is neither convex nor concave in general. The cost of a flow on a wind farm graph is then given by

$$\text{cost}(f) = \sum_{e \in E} c(|f(e)|) \cdot \text{len}(e). \quad (5)$$

The value of $c(|f(e)|)$ stands for the cost per unit length of the cheapest cable type with sufficient capacity to transmit $|f(e)|$ units of turbine production. With all that, WCP is the problem of finding a feasible flow f on a given wind farm graph that minimizes the cost. There is an analogon to the linear-cost integer flow theorem (e. g. [2, Thm. 9.10]) that guarantees an optimal flow with integral values.

► **Lemma 1.** *Suppose the cost function is discontinuous only at integers and there is a feasible flow. Then, there is a cost-minimal integral flow.*

4 Algorithm

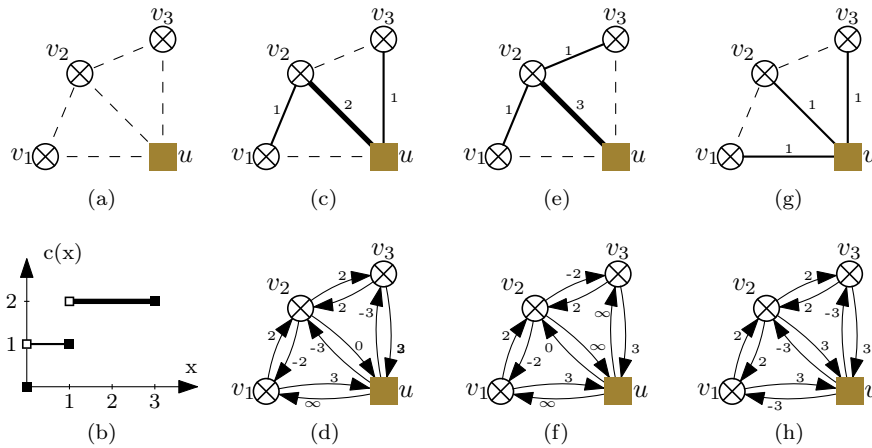
Given a wind farm graph G we define the residual graph R of G with vertices $V(R)$ and edges $E(R)$ by $V(R) = V(G) \cup \{s\}$ and $E(R) = \{e, \bar{e} : e \in E(G)\} \cup \{(v, s), (s, v) : v \in V_S\}$ where \bar{e} is the reverse of e . The new vertex s , the *super substation*, is a virtual substation without capacity, that is connected to all substations. The edges to and from s are used to model the substation capacity constraints and to allow the production of one turbine to be reassigned to another substation.

For a given feasible flow f in G of finite cost and $\Delta \in \mathbb{N}$ we further define *residual costs*, which represent by how much the cost for the edge changes if the flow on the edge is increased by Δ (cf. Figure 1 (a) – (d) for an example). Note that for negative quantities of flow this implies that the absolute value of the flow may be reduced or even the direction of the flow on an edge may change. More formally, we define $\gamma: E(R) \rightarrow \mathbb{R}$ by $\gamma(e) = (c(|f(e) + \Delta|) - c(|f(e)|)) \cdot \text{len}(e)$ for all $e \in E(R)$ that are neither incident to s nor lead to a substation where we alias $f(\bar{e}) = -f(e)$ for all $e \in E(G)$. By this definition the residual costs are infinite if $c(|f(e) + \Delta|) = \infty$, i. e., if the maximal capacity on e is exceeded. For $u \in V_S$ and $v \in V_T$, we set $\gamma(u, v) = \infty$ whenever $f(v, u) < \Delta$ because sending $f(u, v) + \Delta$ units from u to v would otherwise imply that flow leaves a substation. On edges into s , we set $\gamma(u, s) = 0$ if and only if $f(u, s) + \Delta \leq \text{cap}_{\text{sub}}(u)$ and $\gamma(u, s) = \infty$ otherwise. On edges leaving the super substation, we set $\gamma(s, u) = 0$ if and only if $f(u, s) \geq \Delta$ and $\gamma(s, u) = \infty$ otherwise to prevent flow from leaving the substation.

In a nutshell, the Negative Cycle Canceling (NCC) algorithm (Algorithm 1) starts with an initial feasible flow and some value of Δ , computes the residual costs, and looks for a negative cycle¹. If the algorithm finds a negative cycle, it cancels the cycle, i. e., it changes the flow by adding Δ units of flow on all (residual) edges of the cycle. Note that this may decrease the actual amount of flow on edges of G . Then this procedure is repeated with the new flow and a (possibly new) value of Δ . If no negative cycle is found, a new value of Δ is chosen and new residual costs are computed. This loop is repeated until all sensible values of Δ have been considered for a single flow, which is then returned by the algorithm. This flow is of integer-value, since the initial flow is designed to only have integer values and we solely consider natural values for Δ . Without loss of generality we can restrict ourselves to integer flows according to Lemma 1, even though our algorithm does not necessarily find an optimal solution of WCP. One question we answer is to what extent the algorithm benefits from different initial flows and different orders in which the values of Δ are chosen. We present different strategies in Sections 4.3 and 4.4.

The details of the algorithm (cf. Sections 4.1 and 4.2) address problems that arise from the special structure of WCP, namely the non-linear cost function c . Firstly, in classical min-cost flow problems, when c is linear, the cost for changing flow by a certain amount is proportional to the amount of flow change (and the length of the respective edge) and does not depend on the current amount of flow on that edge. Hence, there is no need for computing residual costs for different values of Δ . Secondly, *short* cycles, i. e., cycles of two edges, may have non-zero total cost in WCP (cf. cycle uv_2u in Figure 1 (d)). Canceling such a cycle, however, does not change the flow and therefore does not improve the solution. Hence, only cycles of at least three edges (*long* cycles) are interesting to us because they do not contain both an edge

¹ A *cycle* is a sequence of consecutive edges such that the first edge starts at the same vertex where the last edge ends and such that no two edges start at the same vertex. That is, all cycles are simple. A cycle is said to be *negative* if the sum of residual costs over all edges is negative.



■ **Figure 1** Examples of flows and corresponding residual graphs. (a) shows a wind farm graph. Edges between turbines are of length 2, edges between the substation u and any turbine are of length 3. (b) depicts a cost function induced by two cable types. (c) displays a feasible flow. Dashed lines do not carry any flow. The thickness of solid lines represents the necessary cable type to carry the respective flow. (d) is the residual graph for the flow in (c) and $\Delta = 1$. The super substation is omitted for ease of presentation. There are three negative cycles: uv_2u , uv_2v_1u , and uv_3v_2u in (c). (e) shows the flow obtained by sending one unit of flow along uv_3v_2u in (c). (f) is the residual graph for (e) and $\Delta = 1$. (g) depicts the flow obtained by sending one unit of flow along uv_2v_1u in (c). (h) displays the residual graph for (g) and $\Delta = 1$.

and its reverse. Finding any negative cycle can be done in polynomial time but finding long negative cycles is \mathcal{NP} -hard for general directed graphs [15, Theorem 4 for $k = 3$]. Thirdly, the order of canceling cycles matters (Figure 1 (c) – (g)). In (d), there are two long negative cycles: uv_2v_1u and uv_3v_2u . After canceling uv_3v_2u (Figure 1 (e)), the other cycle uv_2v_1u is not negative anymore. Ultimately, Figure 1 (e) and (f) show that the non-existence of negative cycles in (all) residual graphs does not imply that the underlying flow is optimal – contrary to min-cost flow problems with linear cost functions. In other words, there are flows that represent local but not global minima.

4.1 Detecting Long Negative Cycles

We assume that the reader is familiar with the standard Bellman-Ford algorithm [3, 9], which is a common approach to finding negative cycles. We observed in preliminary experiments that it mostly reports short cycles even if long cycles exist. The reason is that negative residual costs on an edge are repeatedly used if the cost of the reverse edge is, say, zero. In that case, the negative residual cost strongly influences the distance labels on close vertices and overshadows long cycles (see cycle uv_2u in comparison to cycle uv_2v_1u in Figure 1 (d)).

One solution is to prohibit propagating the residual cost of an edge over its reverse edge. To this end, we employ the Bellman-Ford algorithm on the subgraph L of the directed line graph² of R which we obtain from the line graph by removing all edges representing U-turns, i. e., edges of the form (e, \bar{e}) for $e \in E(R)$. We define the cost of an edge (e_1, e_2) in L as $\gamma(e_2)$. At every vertex e of L we maintain a distance label $\ell(e)$ initialized as $\gamma(e)$. Thus, throughout

² The *line graph* $L(G)$ of a directed graph G shows which edges are incident to each other. It is defined by $V(L(G)) = E(G)$ and $E(L(G)) = \{((u, v), (v, w)) : (u, v), (v, w) \in E(G)\}$.

the Bellman-Ford algorithm, $\ell(e)$ represents the length of some walk³ in L starting at any vertex of L and ending at e . By construction of L , the label $\ell(e)$ also stands for some walk in R which ends at the target vertex of e and which does not traverse an edge of R directly after its reverse. Consequently, a cycle C in L corresponds to a closed walk W without U-turns of the same cost in R . In particular, W is not a short cycle, which is what we wanted. It may still occur, however, that W includes an edge and its reverse. In that case, W consists of more cycles that may be negative themselves. Therefore, we decompose the closed walk W into cycles, which, in turn, can be canceled one after another. For more details, refer to Section 4.2.

A downside of running the Bellman-Ford algorithm on the line graph is that more labels have to be stored and the running time of the algorithm is in $\mathcal{O}(|V(L)| \cdot |E(L)|)$, which is worse than the running time on R . We present how to implement an algorithm that directly works on R , that is equivalent to the Bellman-Ford algorithm on L , and that has the asymptotic running time as the original Bellman-Ford algorithm on R . When running the Bellman-Ford algorithm on L , there is one label for every vertex of L . Each of those labels gives rise to a label on an edge of R . The labels at incoming edges of $v \in V(R)$ are used to compute the labels at outgoing edges of v . Let (v, w) and (v, x) be two edges leaving v . Let us assume that (x, v) has the smallest label of all edges entering v . Then, (x, v) is used to relax (v, w) . But it cannot be used to relax (v, x) . To do so, we need the second smallest label of all edges entering v . This yields the following observation.

► **Observation 2.** *For each vertex v of R only the two smallest labels of incoming edges of v are required to correctly update the labels on outgoing edges of v .*

Consequently, throughout our modified version of the Bellman-Ford algorithm, we maintain two distance labels $\ell_1(v)$ and $\ell_2(v)$, and two parent pointers $\text{parent}_1(v)$ and $\text{parent}_2(v)$ for every $v \in V(R)$, respectively. As above, $\ell_i(v)$ with $i = 1, 2$ stands for the length of a U-turn-free walk whose first edge is arbitrary and whose last edge is $(\text{parent}_i(v), v)$. That means that the parent pointers hold the edges that have been used to build the values of the distance labels. The algorithm ensures that $\text{parent}_1(v) \neq \text{parent}_2(v)$ and $\ell_1(v) \leq \ell_2(v)$ for every $v \in V(R)$. If, during a relaxation step, several incumbent labels and a newly computed candidate label have the same value, we break ties in favor of the older labels – as in the original algorithm. With Observation 2 we show reduced bounds for the number of iterations and the overall running time compared to a straightforward implementation on L .

► **Theorem 3.** *If after $2 \cdot |V(R)|$ iterations there is an edge whose label can still be reduced, then there is a negative cycle in L .*

► **Corollary 4.** *A negative cycle in L can be computed in $\mathcal{O}(|V(R)| \cdot |E(R)|)$ time if one exists.*

4.2 Algorithm in Detail

The previously described Bellman-Ford algorithm on L is encapsulated in Algorithm 1. We first compute some initial flow (line 1) using one of eight initialization strategies presented in Section 4.3. In line 3 we compute the residual graph R using a given flow f and a given Δ and run the modified Bellman-Ford algorithm (line 4). In the repeat-loop, we consider one edge after another and check in line 7 if it can be relaxed (again) after the Bellman-Ford algorithm. In that case, we extract a walk W in R with negative costs leading to that edge

³ A *walk* is a sequence of consecutive edges. A walk is called *closed* if the start vertex of the first edge equals the target vertex of the last edge. In particular, every cycle is a closed walk.

by traversing parent pointers. However, canceling W directly may not always improve the costs of the flow as W may still contain an edge and its reverse. We decompose W into a set of simple cycles $\mathcal{C} = \{C_1, \dots, C_k\}$ in line 8 and cancel each cycle independently if it is long and has negative costs (lines 9 to 12). Note that even though W has negative costs, it may happen that only short cycles in \mathcal{C} have negative costs and all long cycles have non-negative costs. In this case we search for another negative cycle in L (line 13).

If no negative cycle in the current graph L is canceled, a new value for Δ is determined according to the *delta strategy* (cf. Section 4.4) in line 14 and new residual costs γ are computed. Line 14 also checks if every possible value for Δ has been used after the last update of f without improving the solution. If so, f is returned.

We apply two well-known speed-up techniques to the Bellman-Ford algorithm. Firstly, if one iteration does not yield any update of any label, then the computation is aborted and no negative cycle can be found in the current residual graph. Secondly, after sorting edges by start vertices, we track whether the labels at a vertex v have been updated since last considering its outgoing edges. If not, then there is no need to relax the outgoing edges.

■ **Algorithm 1** Negative Cycle Canceling.

```

Input: Graph  $G$ , costs  $c$ , edge lengths  $len$ 
Result: A feasible flow  $f$  in  $G$ 
1  $f := \text{InitializeFlow}(G, len)$ ,  $\Delta := \text{InitialDelta}$ 
2 while  $\Delta \neq \text{NULL}$  do
3    $(R, \gamma) := \text{ComputeResidualGraph}(G, c, f, \Delta)$ 
4    $\text{RunBellmanFord}(R, \gamma)$ 
5    $\text{found} := \text{false}$ 
6   foreach  $e \in E(R)$  do
7      $W := \text{FindNegativeClosedWalk}(R, e)$ 
8      $\mathcal{C} := \text{DecomposeWalkIntoCycles}(W)$ 
9     foreach  $C \in \mathcal{C}$  do
10      if  $|C| \geq 3$  and  $\gamma(C) < 0$  then
11         $f := \text{AddFlowOnCycle}(f, C, \Delta)$ 
12         $\text{found} := \text{true}$ 
13      if  $\text{found}$  then break
14    $\Delta := \text{NextDelta}(\Delta, \text{found})$ 
15 return  $f$ 

```

4.3 Initialization Strategies

Before we can start searching for and canceling negative cycles, we need some feasible initial flow. To obtain such a flow, we consider eight strategies, which all roughly work as follows. We pick a turbine u whose production has not been routed to a substation yet. We then search for a shortest path P from u to a substation v with free capacity using Dijkstra's algorithm [8]. The search only considers edges on which the production of the turbine can be routed, i. e., it ignores congested edges. We then route the production of u along P to v .

We consider two metrics to compute shortest paths. Either we use the lengths defined by len (cf. Section 3) or we assume a length of 1 for every edge. Turbine production can either be routed to a nearest or a farthest (in the sense of the respective metric) substation with

free capacity. There are two ways in which the flow is updated: The simpler variant routes only the production of u along P , i. e., the flow along P is increased by 1. The other variant greedily collects as much production from u and other turbines on P as possible without violating any capacity constraints. The resulting flows are integral since the substation capacities and the maximum cable capacity are natural numbers. If no feasible flow of finite cost is found during the initialization, the algorithm returns without a result.

This yields eight initialization strategies, which we name as follows. The base part of each name is either **BFS** if unit distances are used or **Dijkstra** (abbr. **Dijk**) if the distances given by len are used. This part is followed by a suffix specifying the target substation: **Any** (abbr. **A**) for the nearest and **Last** (abbr. **L**) for the farthest substation. An optional prefix of **Collecting** (abbr. **C**) means that the production is greedily collected along shortest paths. For example, **CollectingDijkstraLast** (abbr. **C-Dijk-L**) iterates over all turbines and for each turbine u it finds the substation v such that the shortest path given by len from u to v is longest among all substations. Along a shortest path from u to v , turbine production is collected greedily.

4.4 Delta Strategies

A delta strategy consists of two parts: an initial value for Δ and a function that returns the value of Δ for the following iteration. We discuss eight delta strategies. The simplest one starts with $\Delta = 1$ and increments Δ until a negative cycle is canceled. Then, Δ is reset to 1. We call this strategy **Inc** (as in increasing). Similarly, **Dec** (as in decreasing) starts with the largest possible value for Δ , which is twice the largest cable capacity. Then, Δ is decremented until a cycle is canceled and reset to the largest value. The third strategy **IncDec** behaves like **Inc** until a negative cycle is canceled. Then, it decrements Δ until $\Delta = 1$ and behaves like **Inc** again. To improve performance, all Δ can be skipped during incrementation up to the last value of Δ for which a negative cycle was canceled. The fourth strategy **Random** returns random natural numbers between one and the maximum possible value for Δ . Between any two cycle cancellations, no value is repeated.

For each strategy, we consider the following modification: After canceling a negative cycle, we retain the current value of Δ , recompute the residual costs with the new flow, and run the Bellman-Ford algorithm again. We repeat this, until Δ does not yield a negative cycle. In that case, Δ is changed according to the respective delta strategy. We call the strategies after the modification **StayInc**, **StayDec**, **StayIncDec**, and **StayRandom** (or **S-Inc**, **S-Dec**, **S-IncDec**, and **S-Random** for short).

5 Experimental Evaluation

In the previous sections, we introduce a heuristic with various strategies for the WCP. We first use statistical tests to evaluate these strategies and identify the best ones. Using the result we compare the best *variant* (i. e., best combination of initialization and delta strategy) with different base line algorithms for the WCP namely solving an exact MILP formulation and a Simulated Annealing algorithm [17]. In preliminary experiments a comparison of the MILP solvers Gurobi and CPLEX shows that Gurobi tends to produce better solutions. We therefore use Gurobi to solve the MILP formulation. However, getting an optimal solution takes too long in almost all instances. Thus, we restrict Gurobi to different maximum running times.

For our evaluation we use benchmark sets for wind farms from the literature [17] consisting of wind farms of different sizes and characteristics: small wind farms with exactly one substation (\mathcal{N}_1 : 10–79 turbines), wind farms with multiple substations (\mathcal{N}_2 : 20–79 turbines,

■ **Table 1** Comparison of delta strategies over all initialization strategies. An entry in row i and column j shows on how many instances strategy i produces better solutions than strategy j . Values are marked by a star if they are significant with $p < 10^{-2}$ and by two stars if $p < 10^{-4}$. The best strategy is marked in green.

	Inc	Dec	IncDec	Random	S-Inc	S-Dec	S-IncDec	S-Random
Inc	—	64.1% ^{**}	46.2%	58.1% ^{**}	53.5%	64.0% ^{**}	52.0%	56.9% [*]
Dec	35.9%	—	34.3%	43.1%	37.1%	49.2%	35.1%	39.6%
IncDec	53.8%	65.7% ^{**}	—	59.1% ^{**}	55.2%	64.3% ^{**}	52.7%	57.9% ^{**}
Random	41.9%	56.9% [*]	40.9%	—	46.9%	56.3% [*]	44.1%	44.8%
S-Inc	46.5%	62.9% ^{**}	44.8%	53.1%	—	59.1% ^{**}	46.4%	52.9%
S-Dec	36.0%	50.8%	35.7%	43.7%	40.9%	—	38.6%	42.1%
S-IncDec	48.0%	64.9% ^{**}	47.3%	55.9%	53.6%	61.4% ^{**}	—	55.4%
S-Random	43.1%	60.4% ^{**}	42.1%	55.2%	47.1%	57.9% ^{**}	44.6%	—

\mathcal{N}_3 : 80–180 turbines, \mathcal{N}_4 : 200–499 turbines), and complete graphs (\mathcal{N}_5 : 80–180 turbines). Our code is written in C++14 and compiled with GCC 7.3.1 using the `-O3 -march=native` flags. All simulations are run on a 64-bit architecture with four 12-core CPUs of AMD clocked at 2.1 GHz with 256 GB RAM running OpenSUSE Leap 15.0. We use Gurobi 8.0.0 and all computations are run in single-threaded mode to ensure comparability of the different algorithms.

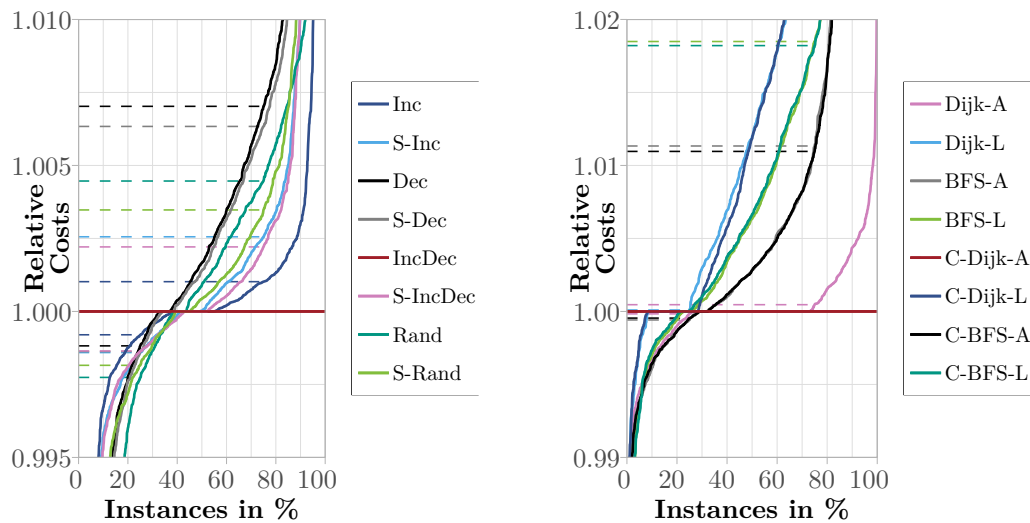
5.1 Comparing Variants of our Algorithm

In a first step, we want to determine which delta strategy works best. To this end, we randomly select 200 instances per benchmark set. We run our algorithm on each instance with every pair of delta and initialization strategy. We first observe that all variants are fast, with running times between tenths of milliseconds to 4.5 minutes on large instances in the worst case. Since all variants have similarly small running times, we base our decision which variant to choose solely on their solution qualities.

To compare the variants in terms of solution quality, we compute for each delta strategy i and instance m the mean $X_m^{(i)}$ of the solution values over all eight initialization strategies. This gives us 1000 data points per delta strategy. For delta strategies i, j we perform a Binomial Sign Test counting instances with $X_m^{(i)} < X_m^{(j)}$ and $X_m^{(j)} < X_m^{(i)}$, that means for this test we are rather interested in whether strategy i performs better than strategy j on instance m and not by how much i is better than j on m . Table 1 summarizes the results of all tests after Bonferroni-correction by 112 (the number of tests from both delta and initialization strategies). The percentage given in an entry in row i and column j states on how many instances i performs strictly better than j after averaging over all initialization strategies. Note that entries (i, j) and (j, i) need not represent 1000 instances, as two variants may return equal solution values.

In the row **IncDec**, all values are above 50%, four of which are significant at the 10^{-4} -level. The smallest value (52.7% in column **StayIncDec**) stands for 481 instances on which **IncDec** performs better than **StayIncDec**. To the contrary, there are 431 instances on which **StayIncDec** yields better solutions (cf. entry 47.3% in row **StayIncDec** and column **IncDec**). While the differences between the four delta strategies involving **Inc** and **IncDec** are not statistically significant, **IncDec** does seem to have a slight advantage over the others. Hence we consider **IncDec** as the best delta strategy.

In Figure 2 (left), for the dark green curve all instances are ordered by $X_m^{(\text{Random})}/X_m^{(\text{IncDec})}$ in ascending order. For a given value α on the abscissa, the curve shows the relative cost factor of the instance at the α -quantile in the computed order. The other curves work



■ **Figure 2** Evaluation of the NCC Algorithm using different strategies. For each strategy and for each instance, the ratio of the best solution value found by that NCC variant to the best solution value found by the reference variant (marked in red) are computed. They are shown in increasing order. The dashed lines represent the 25% and 75% quantiles of the instances. *Left:* The delta strategies are presented relative to the `IncDec` strategy. Solution values represent the average over all initialization strategies. *Right:* The initialization strategies are presented relative to the `CollectingDijkstraAny` strategy with fixed delta strategy `IncDec`.

accordingly. We see, for example, that `IncDec` works strictly better than `StayInc` on 50.9% of all instances and on 7.3% of all instances `IncDec` outperforms `Inc` by at least 0.5% in cost ratio. The minimum ratios range between 0.868 (`Random`) and 0.918 (`StayIncDec`) and the maximum ratios are between 1.069 (`StayDec`) and 1.126 (`StayIncDec`).

Next, we want to find an initialization strategy after fixing `IncDec` as the delta strategy. We pair each initialization strategy with `IncDec` on the same 1000 instances. Table 2 summarizes the results of all pairwise tests after Bonferroni-correction with factor 112. We see that initialization strategies that route turbine production to the nearest substations outperform their counterparts choosing the farthest substations. Among the former, strategies using Euclidean distances work significantly better than strategies involving BFS. In Figure 2 (right) we depict ratios of solution values compared to `CollectingDijkstraAny`. The minimum ratios are between 0.65 and 0.72 for all initialization strategies other than `DijkstraAny`, which is at 0.971. The maximum ratios range between 1.05 and 1.10 for `DijkstraAny`, `BFSLast`, and `CollectingBFSLast` and are around 1.17 for all others. We see that for the main part there is hardly any difference between collecting strategies and their non-collecting counterparts. It stands out that on roughly 40% of all instances `CollectingDijkstraAny` is better than `BFSAny` and `CollectingBFSAny` by 0.5% and better than `BFSLast` and `CollectingBFSLast` by 1%. `CollectingDijkstraAny` has a slight but not significant advantage over `DijkstraAny`. We therefore declare `CollectingDijkstraAny` paired with `IncDec` as our best variant.

Table 3 shows the running time characteristics of `CollectingDijkstraAny` paired with `IncDec`. Running times range between tenths of milliseconds to under two minutes.

■ **Table 2** Comparison of the initialization strategies when the delta strategy `IncDec` is fixed. An entry in row i and column j shows on how many instances strategy i produces better solutions than strategy j . Values are marked by a star if they are significant with $p < 10^{-2}$ and by two stars if $p < 10^{-4}$. The best strategy is marked in green.

	Dijk-A	BFS-A	C-Dijk-A	C-BFS-A	Dijk-L	BFS-L	C-Dijk-L	C-BFS-L
Dijk-A	—	65.5% ^{**}	47.5%	66.8% ^{**}	88.9% ^{**}	76.7% ^{**}	85.3% ^{**}	76.1% ^{**}
BFS-A	34.5%	—	32.7%	50.2%	70.2% ^{**}	66.2% ^{**}	68.5% ^{**}	64.4% ^{**}
C-Dijk-A	52.5%	67.3% ^{**}	—	66.9% ^{**}	88.7% ^{**}	78.1% ^{**}	88.4% ^{**}	77.9% ^{**}
C-BFS-A	33.2%	49.8%	33.1%	—	71.5% ^{**}	64.7% ^{**}	69.6% ^{**}	65.7% ^{**}
Dijk-L	11.1%	29.8%	11.3%	28.5%	—	41.9%	46.7%	42.6%
BFS-L	23.3%	33.8%	21.9%	35.3%	58.1% ^{**}	—	56.6% [*]	51.5%
C-Dijk-L	14.7%	31.5%	11.6%	30.4%	53.3%	43.4%	—	43.5%
C-BFS-L	23.9%	35.6%	22.1%	34.3%	57.4% [*]	48.5%	56.5% [*]	—

■ **Table 3** Minimum, average and maximum of running times in milliseconds of the best NCC variant `IncDec`, `CollectingDijkstraAny`. Running time measurement starts before the initial flow is computed and ends with the termination of the algorithm prior to outputting the solution.

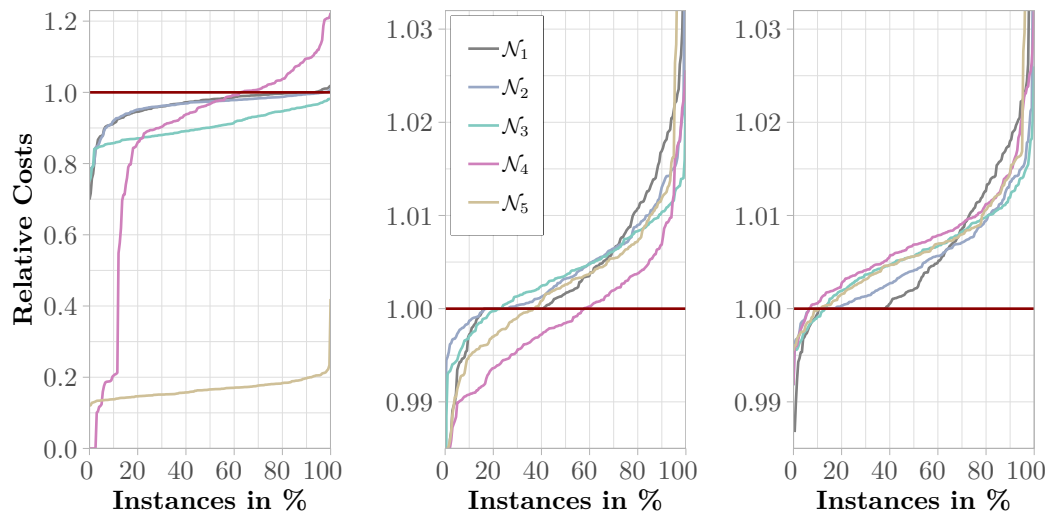
	\mathcal{N}_1	\mathcal{N}_2	\mathcal{N}_3	\mathcal{N}_4	\mathcal{N}_5
min	0.46	3.40	183	3.0k	1.9k
avg	36.0	53.2	717	27.3k	14.0k
max	215	314	3.1k	89.6k	106k

5.2 Comparing our Best Variant with Gurobi

We compare our algorithm in its best variant, i. e., `CollectingDijkstraAny` with `IncDec`, with Gurobi on a MILP formulation which uses a binary variable for each edge and cable type to model the step cost function. We randomly select 200 instances per benchmark set from the benchmark sets in [17].

In Figure 3 we plot the ratio of the best solution value found by our algorithm to Gurobi’s best solution at running times of two seconds, one hour, and one day for each benchmark set separately. As mentioned before, these running times represent both interactive and non-time-critical planning. Since our algorithm terminates in under two minutes, the comparisons in Figure 3 (middle and right) use the solution our algorithm provides at termination. While discussing the plots, we also discuss the so-called relative gaps, a standard notion from MIXED-INTEGER LINEAR PROGRAMMING. From the one-day MILP experiments we know a proven lower bound (lb) on the optimal solution value for each instance. For any instance, any maximum running time and for both the MILP and the NCC algorithm take best solution value (ub) found at the maximum running time and compute $\text{ub-lb}/\text{ub}$. This value is in the unit interval and gives information on how “bad” the solution value can be compared to the (unknown) optimal value. Note, however, that a solution might be optimal even though the gap is positive. We refer to the relative gaps as MILP *gap* and NCC *gap*, respectively.

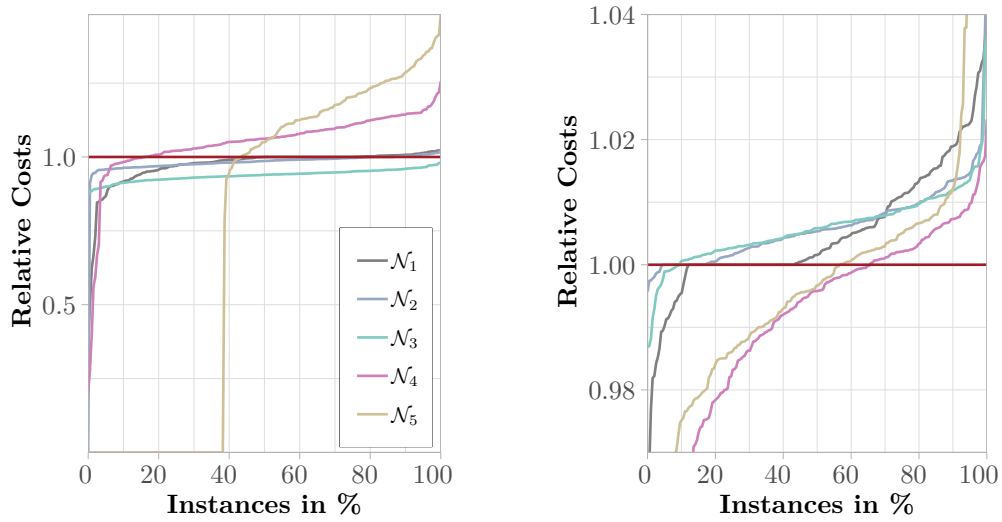
After two seconds our algorithm outperforms Gurobi on all benchmark sets except \mathcal{N}_4 and produces better solutions on 89% of the instances across all benchmark sets. On \mathcal{N}_1 the NCC gaps are on average 14.1% with a maximum of 24.8% compared to MILP gaps of 16.9% on average and at most 43%. For \mathcal{N}_3 , the NCC gaps are on average 37.8% with a spread of only seven percentage points, compared to a mean of 34.6% and a maximum of 45.4% for the MILP gap. The values for \mathcal{N}_2 range between those for \mathcal{N}_1 and \mathcal{N}_3 . The ratios of solution values range between 0.699 and 1.019 for \mathcal{N}_1 , \mathcal{N}_2 , and \mathcal{N}_3 . On \mathcal{N}_4 , which contains the largest instances, our algorithm computes better solutions on 62% of the instances.



■ **Figure 3** Comparison of the NCC algorithm to Gurobi on 200 instances per benchmark set. The ordinate shows the ratio of objective values at various maximum running times of our algorithm to objective values of Gurobi. Running times: *Left*: two seconds, *Middle*: one hour, *Right*: one day.

However, on six instances Gurobi does not find a solution. The instances on which Gurobi is better are on average larger than the other instances in \mathcal{N}_4 . There are 16 instances on which the ratio of solution values exceeds 1.1 with a maximum of 1.223. On those very large instances, detecting negative cycles takes longer and fewer iterations are performed in two seconds. The NCC gaps spread between 31.4% and 57.4% with an average of 42.6%. The MILP gaps are even worse with a mean value of 48.2% and 18 instances above 88.5%. On the complete graphs of \mathcal{N}_5 , our algorithm produces solutions that are at least 75% cheaper than Gurobi's on all but one instance (which has a ratio of 0.420). The gaps, however, are on average at 53.2% for the NCC algorithm and at 99.2% for Gurobi. The large spread in solution values indicates that both Gurobi and our algorithm seem unable to reliably find good solutions within a maximum running time of two seconds.

Within one hour (middle plot in Figure 3) Gurobi finds better solutions than our algorithm on a majority of the instances in all benchmark sets except \mathcal{N}_4 . There our algorithm still yields better solutions in 57.5% of the instances. However, our algorithm is within 1% of Gurobi's best solution on 85.9% and within 2% on 96.8% of all instances. Only on four of 1000 instances (all in \mathcal{N}_5), the ratio exceeds 1.10 with a maximum of 1.203. That means, our algorithm is comparable to Gurobi in solution quality but much faster since it terminates in under two minutes. We make similar observations for running times of one day (right plot in Figure 3). While our algorithm is at least as good as Gurobi only on between 7% (\mathcal{N}_4) and 38.5% (\mathcal{N}_1) of the instances, it is within 1% of Gurobi's solution on 77.6% of all instances. Again, there are only four instances with a ratio worse than 1.10 with a maximum of 1.210. Our algorithm does not profit from long running times since it gets stuck in local minima. Thus, the MILP solver is the better choice if more time is available. Between running times of one hour and day, the gaps look vastly the same and there is hardly any difference between NCC gaps and MILP gaps. They range between zero and 25.0% on \mathcal{N}_1 , clot around 28% for \mathcal{N}_3 and \mathcal{N}_4 and around 34% for \mathcal{N}_5 with seven outliers to the worse by the NCC algorithm.



■ **Figure 4** Comparison of Negative Cycle Canceling algorithm to the Simulated Annealing algorithm on 200 instances per benchmark set. The ordinate represents the ratio of objective values at different maximum running times of our algorithm to objective values of the Simulated Annealing algorithm. *Left*: Running time of two seconds. *Right*: Running time of one hour.

5.3 Comparison to Metaheuristic Simulated Annealing

We compare our best algorithm variant with the best variant of a Simulated Annealing (SA) algorithm [17]. We run the SA algorithm on 200 randomly selected instances per benchmark set (independently selected from other experiments). We compare the best solutions found after two seconds and one hour (Figure 4). After two seconds, our algorithm outperforms the SA algorithm on all instances from \mathcal{N}_3 and on 74.5% and 86.5% on \mathcal{N}_1 and \mathcal{N}_2 , respectively. The minimum ratios are 0.381 for \mathcal{N}_1 and around 0.90 for \mathcal{N}_2 and \mathcal{N}_3 with one instance in \mathcal{N}_2 where the SA algorithm does not find a solution. The maximum ratio on those benchmark sets is at most 1.024. On the larger instances of \mathcal{N}_4 and \mathcal{N}_5 , our algorithm presumably cannot perform sufficient iterations, as the SA algorithm is better on 70% of those instances. Yet, the SA algorithm does not find feasible solutions on 38.5% of instances from \mathcal{N}_5 . The ratios have a wide spread: from 0.203 to 1.256 for \mathcal{N}_4 and 0.788 to 1.482 for \mathcal{N}_5 (save for the instances without a solution from the SA algorithm). After one hour, the SA algorithm provides better solutions than our algorithm on 83% and 90.5% of instances from \mathcal{N}_2 and \mathcal{N}_3 , respectively. Our algorithm, however, stays within 1% in solution quality on 77% on the benchmark sets \mathcal{N}_1 – \mathcal{N}_3 . Again, our algorithm seems to be stuck in local minima. On \mathcal{N}_4 and \mathcal{N}_5 , our algorithm performs better than the SA algorithm on 65.5% and 56.5%, respectively. Apparently, the SA algorithm needs more time to explore the solution space. The minimum ratios of solution values are as low as 0.716 for \mathcal{N}_1 and between 0.908 and 0.996 for the other benchmark sets. The maximum ratios are at most 1.055 for all benchmark sets except \mathcal{N}_5 (1.179). This supports our findings from the MILP experiments that our algorithm is competitive to other approaches to solving WCP within very short amounts of time. In view of an interactive planning process, it stands out that the SA algorithm struggles to find solutions quickly in dense graphs.

6 Conclusion

Based on recently presented ideas [14] we propose and compare numerous variants of a Negative Cycle Canceling heuristic for the WIND FARM CABLING PROBLEM. While all variants run in the order of milliseconds up to 4.5 minutes, they differ significantly in quality. We identify the best variant and use it to compare our heuristic to the MILP solver Gurobi and a Simulated Annealing algorithm from the literature. With these comparisons we are able to solve several open questions [14]. While the MILP solver Gurobi has the potential to find optimal solutions if it runs long enough, our heuristic is able to find solutions of comparable quality in only a fraction of the time. Our algorithm beats Gurobi in finding good solutions in a matter of seconds. We make similar observations when we compare ourselves with a Simulated Annealing approach.

Moving forward, one may investigate how to improve the solution quality of our heuristic. Visually comparing flows from our algorithm and other solution methods may help to identify what kind of more complex circulations improve the solution. It then remains to investigate how these circulations can be detected. Also, methods for escaping local minima such as temporarily allowing worse solutions could help to improve our algorithm. It also remains open whether one can prove any theoretical guarantees on the solution quality or the number of iterations. Along the same lines, any theoretical insights on why one delta or initialization strategy works better than another, or on the order in which cycles should be canceled could help improve the NCC algorithm.

In a broader algorithmic view, the heuristic can be easily generalized to minimum-cost flow problems with other types of cost functions provided that one searches for integral flows. It would be interesting to see how well the heuristic performs there.

References

- 1 4C Offshore Ltd. *Hornsea Project Three Offshore Wind Farm*, 2018. Accessed: 2018-08-15. URL: www.4c offshore.com/windfarms/hornsea-project-three-united-kingdom-uk1k.html.
- 2 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, Upper Saddle River, NJ [u.a.], 1993.
- 3 Richard Bellman. On a Routing Problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958. doi:10.1090/qam/102435.
- 4 Constantin Berzan, Kalyan Veeramachaneni, James McDermott, and Una-May O’Reilly. Algorithms for cable network design on large-scale wind farms. MSRP technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 2011. http://thirld.com/files/msrp_techreport.pdf, Accessed: 2016-01-04.
- 5 Boris V. Cherkassky and Andrew V. Goldberg. Negative-cycle detection algorithms. *Mathematical Programming*, 85(2):277–311, June 1999. doi:10.1007/s101070050058.
- 6 Ouahid Dahmani, Salvy Bourguet, Mohamed Machmoum, Patrick Guerin, Pauline Rhein, and Lionel Josse. Optimization of the Connection Topology of an Offshore Wind Farm Network. *IEEE Systems Journal*, 9(4):1519–1528, 2015. doi:10.1109/JSYST.2014.2330064.
- 7 Mauricio C. de Souza, Philippe Mahey, and Bernard Gendron. Cycle-based algorithms for multicommodity network flow problems with separable piecewise convex costs. *Networks*, 51(2):133–141, 2008. doi:10.1002/net.20208.
- 8 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959. doi:10.1007/BF01386390.
- 9 Lester R. Ford, Jr. and Delbert R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, USA, 2010.

- 10 Virginie Gabrel, Arnaud Knippel, and Michel Minoux. Exact solution of multicommodity network optimization problems with general step cost functions. *Operations Research Letters*, 25(1):15–23, 1999. doi:10.1016/S0167-6377(99)00020-6.
- 11 Andrew V. Goldberg and Tomasz Radzik. A heuristic improvement of the Bellman-Ford algorithm. *Applied Mathematics Letters*, 6(3):3–6, 1993. doi:10.1016/0893-9659(93)90022-F.
- 12 Andrew V. Goldberg and Robert E. Tarjan. Finding Minimum-cost Circulations by Canceling Negative Cycles. *Journal of the ACM*, 36(4):873–886, October 1989. doi:10.1145/76359.76368.
- 13 Donald Goldfarb, Jianxiu Hao, and Sheng-Roan Kai. Shortest Path Algorithms Using Dynamic Breadth-First Search. *Networks*, 21(1):29–50, 1991. doi:10.1002/net.3230210105.
- 14 Sascha Gritzbach, Torsten Ueckerdt, Dorothea Wagner, Franziska Wegner, and Matthias Wolf. Towards negative cycle canceling in wind farm cable layout optimization. In *Proceedings of the 7th DACH+ Conference on Energy Informatics*, volume 1 (Suppl 1). Springer, 2018. doi:10.1186/s42162-018-0030-6.
- 15 Longkun Guo and Peng Li. On the Complexity of Detecting k -Length Negative Cost Cycles. In *Combinatorial Optimization and Applications, COCOA 2017*, volume 10627 of *Lecture Notes in Computer Science*, pages 240–250. Springer International Publishing, 2017. doi:10.1007/978-3-319-71150-8_21.
- 16 Morton Klein. A Primal Method for Minimal Cost Flows with Applications to the Assignment and Transportation Problems. *Management Science*, 14(3):205–220, 1967. doi:10.1287/mnsc.14.3.205.
- 17 Sebastian Lehmann, Ignaz Rutter, Dorothea Wagner, and Franziska Wegner. A Simulated-Annealing-Based Approach for Wind Farm Cabling. In *Proceedings of the Eighth International Conference on Future Energy Systems, e-Energy '17*, pages 203–215, New York, NY, USA, 2017. ACM. doi:10.1145/3077839.3077843.
- 18 Sara Lumbreras and Andres Ramos. Optimal Design of the Electrical Layout of an Offshore Wind Farm Applying Decomposition Strategies. *IEEE Transactions on Power Systems*, 28(2):1434–1441, 2013. doi:10.1109/TPWRS.2012.2204906.
- 19 New York State Energy Research and Development Authority. *New York State Offshore Wind Master Plan*, 2017. Accessed: 2018-08-15. URL: <https://www.nyserda.ny.gov/-/media/Files/Publications/Research/Biomass-Solar-Wind/Master-Plan/Offshore-Wind-Master-Plan.pdf>.
- 20 Adam Ouorou and Philippe Mahey. A minimum mean cycle cancelling method for nonlinear multicommodity flow problems. *European Journal of Operational Research*, 121(3):532–548, 2000. doi:10.1016/S0377-2217(99)00050-8.
- 21 Christos H. Papadimitriou. The complexity of the capacitated tree problem. *Networks*, 8(3):217–230, 1978. doi:10.1002/net.3230080306.
- 22 Tomasz Radzik and Andrew V. Goldberg. Tight bounds on the number of minimum-mean cycle cancellations and related results. *Algorithmica*, 11(3):226–242, March 1994. doi:10.1007/BF01240734.
- 23 Pedro Santos Valverde, António J. N. A. Sarmento, and Marco Alves. Offshore Wind Farm Layout Optimization – State of the Art. *Journal of Ocean and Wind Energy*, 1(1):23–29, 2014.
- 24 WindEurope asbl/vzw. *Wind in power 2017*, 2018. Accessed: 2018-08-15. URL: <https://windeurope.org/wp-content/uploads/files/about-wind/statistics/WindEurope-Annual-Statistics-2017.pdf>.
- 25 Menghua Zhao, Zhe Chen, and Frede Blaabjerg. Optimization of Electrical System for a Large DC Offshore Wind Farm by Genetic Algorithm. In *Proceedings of NORPIE 2004*, pages 1–8, 2004.

Towards Improving Christofides Algorithm for Half-Integer TSP

Arash Haddadan

Carnegie Mellon University, Pittsburgh, PA, USA
ahaddada@andrew.cmu.edu

Alantha Newman

CNRS, Université Grenoble Alpes, Grenoble, France
alantha.newman@grenoble-inp.fr

Abstract

We study the traveling salesman problem (TSP) in the case when the objective function of the subtour linear programming relaxation is minimized by a *half-cycle point*: $x_e \in \{0, \frac{1}{2}, 1\}$ where the half-edges form a 2-factor and the 1-edges form a perfect matching. Such points are sufficient to resolve half-integer TSP in general and they have been conjectured to demonstrate the largest integrality gap for the subtour relaxation.

For half-cycle points, the best-known approximation guarantee is $\frac{3}{2}$ due to Christofides' famous algorithm. Proving an integrality gap of α for the subtour relaxation is equivalent to showing that αx can be written as a convex combination of tours, where x is any feasible solution for this relaxation. To beat Christofides' bound, our goal is to show that $(\frac{3}{2} - \epsilon)x$ can be written as a convex combination of tours for some positive constant ϵ . Let $y_e = \frac{3}{2} - \epsilon$ when $x_e = 1$ and $y_e = \frac{3}{4}$ when $x_e = \frac{1}{2}$. As a first step towards this goal, our main result is to show that y can be written as a convex combination of tours. In other words, we show that we can *save on 1-edges*, which has several applications. Among them, it gives an alternative algorithm for the recently studied *uniform cover* problem. Our main new technique is a procedure to glue tours over proper 3-edge cuts that are tight with respect to x , thus reducing the problem to a base case in which such cuts do not occur.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization

Keywords and phrases Traveling salesman problem, subtour elimination relaxation, integrality gap, gluing subtours

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.56

Related Version A full version of the paper is available at: <https://arxiv.org/pdf/1907.02120.pdf> [12].

Funding *Arash Haddadan*: Supported in part by ONR grant N00014-18-1-2099 and NSF grant CCF-1527032.

Alantha Newman: Supported in part by IDEX-IRS SACRE.

Acknowledgements We thank R. Ravi for comments on the presentation of this paper.

1 Introduction

In the **traveling salesman problem (TSP)** we are given a complete graph $G = (V, E)$ together with a vector $c \in \mathbb{R}_{\geq 0}^E$ of edge costs satisfying the triangle inequality: $c_{uv} + c_{vw} \geq c_{uw}$ for $u, v, w \in V$. The goal is to find a minimum cost Hamiltonian cycle of G . The following formulation is a classic linear programming relaxation for TSP [9].

$$\min \left\{ \sum_{e \in E} c_e x_e : \sum_{u \in V \setminus \{v\}} x_{vu} = 2 \text{ for } v \in V, \sum_{v \in S, u \notin S} x_{vu} \geq 2 \text{ for } \emptyset \subset S \subset V, x \in \mathbb{R}_{\geq 0}^E \right\}.$$



© Arash Haddadan and Alantha Newman;

licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 56; pp. 56:1–56:12

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Let $\text{SUBTOUR}(G)$ denote the feasible region of this relaxation. We will refer to $\sum_{e \in E} c_e x_e$ as the *objective function*. A *tour* of G is a connected, spanning, Eulerian multi-subgraph of G . It is well known that due to the triangle inequality on the edge costs, a tour of G can be turned into a Hamiltonian cycle of G of no greater cost. For any $x \in \text{SUBTOUR}(G)$, the vector $\frac{3}{2}x$ can be decomposed into a convex combination of tours of G . This follows from a polyhedral analysis of Christofides' famous $\frac{3}{2}$ -approximation algorithm [7, 21, 20]. For a point $x \in \text{SUBTOUR}(G)$, define $G_x = (V, E_x = \{e \in E : x_e > 0\})$ to be the support graph of x . Let $\text{TSP}(G_x)$ be the convex hull of characteristic vectors of tours of G_x . The following conjecture is well-known and widely studied and implies a $\frac{4}{3}$ -approximation algorithm for TSP.

► **Conjecture 1** (The Four-Thirds Conjecture). *If $x \in \text{SUBTOUR}(G)$, then $\frac{4}{3}x \in \text{TSP}(G_x)$.*

However, more than four decades after the publication of Christofides' algorithm, there is still no $(\frac{3}{2} - \epsilon)$ -approximation algorithm known for TSP. For special cases, there has been some progress in the past few years. For example, in the *unweighted* case where the edge costs correspond to the shortest path metric of an unweighted graph, a series of papers improved the $\frac{3}{2}$ factor to $\frac{7}{5}$ [16, 15, 19].

One interesting special case of *weighted* TSP is when the solution $x \in \text{SUBTOUR}(G)$ that minimizes the objective function is half-integer. In the unweighted case, if a half-integer point $x \in \text{SUBTOUR}(G)$ minimizes the objective function, then there is a $\frac{4}{3}$ -approximation algorithm for TSP [15].

► **Problem 2** (Half-integer TSP). *For $x \in \text{SUBTOUR}(G) \cap \{0, \frac{1}{2}, 1\}^E$, henceforth a half-integer point, show $\alpha x \in \text{TSP}(G_x)$ for constant $\alpha \in [1, \frac{3}{2})$.*

Consider a half-integer point $x \in \text{SUBTOUR}(G) \cap \{0, \frac{1}{2}, 1\}^E$ and let $H_x = \{e \in E : x_e = \frac{1}{2}\}$ and $W_x = \{e \in E : x_e = 1\}$. Carr and Vempala showed that in Problem 2, we can assume without loss of generality a stronger condition for $x \in \text{SUBTOUR}(G)$: a *half-integer Carr-Vempala point* is a half-integer point such that the support graph G_x is a cubic graph and for every vertex $u \in V$, there is exactly one edge e incident on u with $x_e = 1$ and two edges f, g incident on u with $x_f = x_g = \frac{1}{2}$. Moreover, H_x forms a Hamilton cycle of G_x , and W_x forms a perfect matching of G_x . If for any half-integer Carr-Vempala point x we have $\alpha x \in \text{TSP}(G_x)$, then for any half-integer point y we have $\alpha y \in \text{TSP}(G_y)$ [6, 4].

We consider a generalization of a half-integer Carr-Vempala point called a *half-cycle point*, which is a half-integer point $x \in \text{SUBTOUR}(G)$ such that the graph G_x is a cubic graph and for every vertex $u \in V$, there is exactly one edge e incident on u with $x_e = 1$ and two edges f, g incident on u with $x_f = x_g = \frac{1}{2}$. This implies that H_x , the half-edges in G_x , forms a 2-factor of G (in which the minimum cycle length is three). Formally, we define a half-cycle point as follows.

► **Definition 3.** *A vector $x \in \text{SUBTOUR}(G)$ is called a half-cycle point if the support graph G_x of x is cubic and 2-edge-connected and $x_e \in \{1, \frac{1}{2}\}$ for all $e \in E_x$.*

Half-cycle points have been studied in restricted cases when all cycles in the 2-factor are triangles [2, 3] or squares [4, 11]. Schalekamp, Williamson and van Zuylen conjectured that the largest gap between $\text{SUBTOUR}(G)$ and $\text{TSP}(G_x)$ occurs for half-cycle points in which the 2-factor consists of odd-cycles [17].¹ We can restate Problem 1 as follows.

¹ Their precise conjecture is that instances of TSP that have an optimal solution $x \in \text{SUBTOUR}(G)$ that is also an optimal *fractional 2-matching* exhibit the largest integrality gap for $\text{SUBTOUR}(G)$. The extreme points of the fractional 2-matching polytope are half-cycle points in which all cycles in the 2-factor are odd [1].

► **Problem 4** (Half-integer TSP). *Let $x \in \mathbb{R}_{\geq 0}^E$ be a half-cycle point. Show $\alpha x \in \text{TSP}(G_x)$ for constant $\alpha \in [1, \frac{3}{2})$.*

We can also state Problem 4 in different way.

► **Problem 5** (Half-integer TSP). *Let $x \in \mathbb{R}_{> 0}^E$ be a half-cycle point. Define vector $y \in \mathbb{R}^{E_x}$ as follows: $y_e = \frac{3}{2} - \epsilon$ for $e \in W_x$ and $y_e = \frac{3}{4} - \delta$ for $e \in H_x$. Show there exists constants $\epsilon, \delta > 0$ such that $y \in \text{TSP}(G_x)$.*

The aforementioned polyhedral analysis of Christofides' algorithm implies the following theorem.

► **Theorem 6** ([7, 21, 20]). *Let $x \in \mathbb{R}_{\geq 0}^E$ be a half-cycle point. Define vector $y \in \mathbb{R}^{E_x}$ as follows: $y_e = \frac{3}{2}$ for $e \in W_x$ and $y_e = \frac{3}{4}$ for $e \in H_x$. Then $y \in \text{TSP}(G_x)$.*

Our main result is the following.

► **Theorem 7**. *Let $x \in \mathbb{R}_{\geq 0}^E$ be a half-cycle point. Define vector $y \in \mathbb{R}^{E_x}$ as follows: $y_e = \frac{3}{2} - \frac{1}{20}$ for $e \in W_x$ and $y_e = \frac{3}{4}$ for $e \in H_x$. Then $y \in \text{TSP}(G_x)$.*

While Theorem 7 is not strong enough to resolve Problem 5 (and therefore Problem 4), it does have several applications. For example, given an edge cost function c for which a half-cycle point $x \in \text{SUBTOUR}(G)$ minimizes the objective function, if the total edge costs of the 1-edges is a constant fraction of the total cost of the half-edges, then by Theorem 7, we obtain an approximation factor better than $\frac{3}{2}$.

Another application is related to the problem of *uniform covers* posed by Sebő [18]. Let x be a *cubic point* if $x \in \text{SUBTOUR}(G) \cap \{0, \frac{2}{3}\}$. Observe that G_x is cubic and 3-edge-connected.

► **Problem 8** (Uniform cover problem). *Let x be a cubic point. Show that $\alpha x \in \text{TSP}(G_x)$ for constant $\alpha \in [1, \frac{3}{2})$.*

Recently, Haddadan, Newman and Ravi gave a positive answer to Problem 8 and showed $\alpha \leq \frac{27}{19} \approx 1.421$ [13]. Previously, Boyd and Sebő had shown that $\alpha \leq \frac{9}{7} \approx 1.286$ if G_x is additionally Hamiltonian [4]. In fact, Theorem 7 gives an alternative way to answer Problem 8.

► **Lemma 9**. *Let x be a half-cycle point. Define vector $y \in \mathbb{R}^{E_x}$ as follows: $y_e = \frac{3}{2} - \epsilon$ for $e \in W_x$ and $y_e = \frac{3}{4} - \delta$ for $e \in H_x$ for constants $\epsilon, \delta \geq 0$. Suppose $y \in \text{TSP}(G_x)$. Then for any cubic point z , we have $\alpha z \in \text{TSP}(G_z)$ for $\alpha = \frac{3}{2} - \frac{\epsilon}{2} - \delta$.*

In other words, suppose that we can *save either on the 1-edges or on the half-edges*. Then we can solve the uniform cover problem. Moreover, Theorem 7 can be used to slightly improve the currently best-known factors for Problem 8. The proofs of Lemma 9 and Theorem 10 can be found in the full version [12].

► **Theorem 10**. *Let x be a cubic point. Then $\alpha x \in \text{TSP}(G_x)$ for $\alpha = 1.416$. If G_x is Hamiltonian, then $1.279x \in \text{TSP}(G_x)$.*

On a high level, our proof of Theorem 7 is based on Christofides' algorithm: We show that a half-cycle point x can be written as a convex combination of spanning subgraphs with certain properties and then we show that vector $y \in \mathbb{R}^{E_x}$, where $y_e = \frac{9}{20}$ for $e \in W_x$ and $y_e = \frac{1}{4}$ for $e \in H_x$, can be used for parity correction. Our main new tool is a procedure to glue tours over *critical cuts*. For $S \subset V$, let $\delta(S) \subset E_x$ denote the subset of edges crossing the cut $(S, V \setminus S)$.

► **Definition 11.** Let x be a half-cycle point. A proper cut² $S \subset V$ in G_x is called *critical* if $|\delta(S)| = 3$ and $\delta(S)$ contains exactly one edge e with $x_e = 1$. Moreover, for each pair of edges in $\delta(S)$, their endpoints in S (and in $V \setminus S$) are distinct.

Observe that a critical cut in G_x is a proper 3-edge cut that is *tight*: the x -values of the three edges crossing the cut sum to 2. Thus, critical cuts are difficult to handle using an approach based on Christofides' algorithm. In particular, using $(\frac{1}{2} - \epsilon)x$ would be insufficient for parity correction of a critical cut if it is crossed by an odd number of edges in the spanning subgraph.

Applying our gluing procedure, we can reduce TSP on half-cycle points to a problem (i.e., base case) where there are only two types of tight 3-edge cuts. The first type of cut is a *vertex cut*, which we show are easier to handle. In particular, the parity of vertex cuts can be addressed with a key tool used by Boyd and Sebö [4] called *rainbow v -trees* (see Theorem 17). We refer to the second type of cut as a *degenerate tight cut*, which is a cut $S \subset V$ such that $|\delta(S)| = 3$, $|S| > 3$ and $|V \setminus S| > 3$ and the two half-edges in $\delta(S)$ share an endpoint in either S or $V \setminus S$. (Observe that for every degenerate tight cut in G_x , there is a 2-edge cut in G_x .) These cuts are also easier to handle. Using this in combination with a decomposition of the 1-edges into few *induced matchings* (see Definition 18), which have some additional required properties, we can prove Theorem 7 for the base case. We discuss gluing procedures in more detail in Section 1.1.

Let us look back at Problem 2. Let x be a *quartic point* if $x \in \text{SUBTOUR}(G) \cap \{0, \frac{1}{2}\}$. Observe that G_x is 4-regular and 4-edge-connected. Yet another equivalent version of Problem 2 is as follows.

► **Problem 12** (Half-integer TSP). Let x be a quartic point. Show $\alpha x \in \text{TSP}(G_x)$ for $\alpha \in [1, \frac{3}{2}]$.

If we assume that the only 4-edge cuts of G_x are its vertex cuts and the number of vertices is even, we can answer this problem.

► **Theorem 13.** Let x be a quartic point. If G_x has an even number of vertices, and G_x does not have any proper 4-edge cuts, then $(\frac{3}{2} - \frac{1}{42})x \in \text{TSP}(G_x)$.

In the case of a quartic point, Theorem 13 could serve as the base case for if we were able to glue over proper 4-edge cuts of G_x . However, the main difference here is that the gluing arguments we presented for half-cycle points can not easily be extended to this case due to the increased complexity of the distribution of patterns. The proof of Theorem 13 can be found in the full version [12].

1.1 Gluing tours over cuts

The approach of gluing solutions over (often) 3-edge cuts and thereby reducing to an instance without such cuts has been used previously for TSP (e.g., [8]) and extensively in the case of two related problems: the **2-edge-connected multigraph problem (2EC)** and the **2-edge-connected subgraph problem (2ECSS)**. In 2EC, we want to find a minimum cost 2-edge-connected spanning multi-subgraph (henceforth, multigraph for brevity), and in 2ECSS, we want to find a minimum cost 2-edge-connected spanning subgraph (i.e., we are not allowed to double edges). Let $2\text{EC}(G_x)$ and $2\text{ECSS}(G_x)$ denote that convex hulls of characteristic vectors of 2-edge-connected multigraphs and subgraphs, respectively, of G_x . Observe that $\text{TSP}(G_x) \subseteq 2\text{EC}(G_x)$ and $2\text{ECSS}(G_x) \subseteq 2\text{EC}(G_x)$.

² A cut $S \subset V$ is *proper* if $|S| \geq 2$ and $|V \setminus S| \geq 2$.

For example, consider the problem of showing $\frac{6}{5}x \in 2\text{ECSS}(G_x)$ for a cubic point x [3]. Here, we can assume that G_x is essentially 4-edge-connected due to the following commonly used observation. Let $S \subset V$ be a subset of vertices such that $|\delta(S)| = 3$ in G_x . We construct graphs, $G_{\bar{S}}$ and G_S by contracting the sets \bar{S} and S , respectively, in G_x to a *pseudovortex*. Suppose that the graphs $G_{\bar{S}}$ and G_S contain no proper 3-edge cuts and suppose we can write αx restricted to the edge set of each graph as a convex combination of 2-edge-connected subgraphs of the respective graph. Let us consider the patterns around the pseudovortices; each vertex can be adjacent to two or three edges and therefore, there are only four possible patterns around a vertex. Moreover, since each pattern appears the same percentage of time (in the respective convex combinations) for each pseudovortex, tours with corresponding patterns can be *glued* over the 3-edge cut. (For a more formal presentation of this argument, see Lemma 3.3 in [11] or Case 2 in Section 3.1.2 in [14].) Thus, for 2ECSS, this gluing procedure is quite straightforward. Gluing has also been used for 2EC, but here it is necessary to make certain extra assumptions to control the number of patterns around a vertex, due to the fact that the distribution of possible patterns is more complex. Carr and Ravi proved that the vector $\frac{4}{3}x \in 2\text{EC}(G_x)$ for a half-integer point x [5]. To control the number of patterns so that they can use gluing, they require some strong assumptions on the multigraphs in their convex combinations: for example, no edge e with $x_e = \frac{1}{2}$ is doubled and some arbitrarily chosen edge is never used.

In contrast, it appears that no such gluing procedure has been used in approximation algorithms for TSP. Indeed, gluing proofs for 2ECSS and 2EC [5, 3, 14] can not be easily extended to TSP for several reasons: (1) As just discussed, they are used for gluing subgraphs (no doubled edges), while for multigraphs, there are often too many different patterns around a vertex. (For TSP, we must allow doubled edges.) (2) They do not necessarily preserve parity of the vertex degrees. Finally, (3) many of the results for 2ECSS and 2EC based on gluing do not result in polynomial-time algorithms.

The main technical contribution of this paper is to show that for a carefully chosen set of tours, we can design a gluing procedure over critical cuts. In particular, we can fix a critical cut $S \subset V$ in G_x and find a convex combination of tours for G_S . Then we can find a set of tours for $G_{\bar{S}}$ such that the distribution of patterns around the pseudovortex corresponding to S matches that of the pseudovortex corresponding to \bar{S} in G_S . This is done by separately matching the pattern for the spanning subgraphs and for the parity correction. In fact, while each vertex may have a different set of patterns around it, we show that the patterns around each vertex can be encapsulated by a single parameter: the fraction of times in the convex combination of spanning subgraphs that a vertex is a leaf. There can be some flexibility in this degree distribution for some arbitrarily chosen vertex, and this is what we exploit to sufficiently control the patterns around a pseudovortex to enable gluing.

1.2 Definitions, tools and notation

► **Definition 14.** Let $G = (V, E)$ be a graph. For a vertex $v \in V$, a *v-tree* of G is a subset F of E such that $|F \cap \delta(v)| = 2$ and $F \setminus \delta(v)$ induces a spanning tree of $V \setminus \{v\}$.

Denote by $v\text{-TREE}(G)$ the convex hull of incidence vectors of *v-trees* of G . The $v\text{-TREE}(G)$ is characterized by the following linear inequalities.

$$\begin{aligned} v\text{-TREE}(G) = \{x \in [0, 1] : x(\delta(v)) = 2, \\ x(E[U]) \leq |U| - 1 \text{ for all } \emptyset \subset U \subseteq V \setminus \{v\}, x(E) = |V|\}. \end{aligned}$$

► **Definition 15.** Let $G = (V, E)$ and v be a vertex of G . Let \mathcal{P} a collection of disjoint subsets of E . A \mathcal{P} -rainbow v -tree of G , is a v -tree of G such that $|T \cap P| = 1$ for $P \in \mathcal{P}$.

► **Definition 16.** Let $G = (V, E)$ and let x be a vector $(0, 1]^E$. Let \mathcal{S} denote a set of subgraphs of G (i.e., each $S \subseteq E$ for each $S \in \mathcal{S}$). If there is a probability distribution $\lambda = \{\lambda_S\}_{S \in \mathcal{S}}$ such that $x = \sum_{S \in \mathcal{S}} \lambda_S \chi^S$, then we say $\{\lambda, \mathcal{S}\}$ is a convex combination for x . If such a probability distribution exists, then we say that x can be decomposed into (or written as) a convex combination of subgraphs in \mathcal{S} .

► **Theorem 17** (Boyd, Sebö [4]). Let $x \in \text{SUBTOUR}(G)$ and \mathcal{P} be a collection of disjoint subsets of E such that $x(P) = 1$ for $P \in \mathcal{P}$. Then, x can be decomposed into a convex combination of \mathcal{P} -rainbow v -trees of G_x for any $v \in V$.

► **Definition 18.** Given a graph $G = (V, E)$, a set of edges $M \subseteq E$ forms an induced matching in G if the subgraph of G induced on the endpoints of M forms a matching (i.e., if edges e and f belong to an induced matching M , then there is no 3-edge path in G containing both e and f).

Consider a half-cycle point x . For a vertex u in G_x we denote by e_u the unique 1-edge incident on u and by $\gamma(u)$ the two vertices that are the other endpoints of the half-edges incident on v . In other words, suppose $\delta(u) = \{e_u, f, g\}$ and suppose that w_1 and w_2 are the other endpoints of f and g , respectively. Then $\gamma(u) = \{w_1, w_2\}$.

2 Saving on 1-edges for half-cycle points

Let x be a half-cycle point. In this section, we present an algorithm to write x as a convex combination of tours of G_x . Following Christofides' algorithm, we first construct a convex combination of spanning subgraphs in Section 2.1. Next, we address parity correction in Section 2.2. We combine these two steps in Section 2.3 for the base case, in which G_x contains no critical cuts. In Section 2.4, we show how to iteratively glue tours for base cases together to construct tours for general G_x .

2.1 Convex combinations of spanning subgraphs

► **Definition 19.** Let x be a half-cycle point and let v be a vertex of G_x . Suppose $M \subset W_x$ is a subset of 1-edges of G_x . Let $0 \leq \Lambda \leq \frac{1}{2}$. Let \mathcal{T} be a set of spanning connected subgraphs of G_x and let $\lambda = \{\lambda_T\}_{T \in \mathcal{T}}$ be a probability distribution such that $\{\lambda, \mathcal{T}\}$ is a convex combination for x . Then we say $P(v, M, \Lambda)$ holds for the convex combination $\{\lambda, \mathcal{T}\}$ if it has the following properties.

1. $\sum_{T \in \mathcal{T}: |\delta_T(v)|=1} \lambda_T = \sum_{T \in \mathcal{T}: |\delta_T(v)|=3} \lambda_T = \Lambda$ and $\sum_{T \in \mathcal{T}: |\delta_T(v)|=2} \lambda_T = 1 - 2\Lambda$.
2. For each edge $st \in M$, $|\delta_T(s)| = |\delta_T(t)| = 2$ for $T \in \mathcal{T}$.
3. $T \setminus \delta_T(v)$ induces a spanning subgraph on $V \setminus \{v\}$.

► **Lemma 20.** Suppose $M \subset W_x$ forms an induced matching in G_x and edge $e_v \in M$. Then there is a set of spanning connected subgraphs \mathcal{T} of G_x and a probability distribution $\lambda = \{\lambda_T\}_{T \in \mathcal{T}}$ such that $\{\lambda, \mathcal{T}\}$ is a convex combination for x for which $P(v, M, 0)$ holds.

Proof. For each $st \in M$, pair the half-edges incident on s and pair those incident on t to obtain disjoint subsets of edges \mathcal{P} and decompose x into a convex combination of \mathcal{P} -rainbow v -trees \mathcal{T} (i.e., $x = \sum_{T \in \mathcal{T}} \lambda_T \chi^T$) via Theorem 17. This is the desired convex combination since for all $T \in \mathcal{T}$, we have $|\delta_T(v)| = 2$ and $|\delta_T(u)| = 2$ for all endpoints u of edges in M . Thus, the first and second conditions are satisfied. The third condition holds by definition of v -trees. ◀

► **Lemma 21.** *Let $\gamma(v) = \{w_1, w_2\}$ and let Λ be any constant such that $0 \leq \Lambda \leq \frac{1}{2}$. If $M \subset W_x$ forms an induced matching in G_x , $e_v \notin M$ and $|M \cap \{e_{w_1}, e_{w_2}\}| \leq 1$. Then there is a set of spanning connected subgraphs \mathcal{T} of G_x and a probability distribution $\lambda = \{\lambda_T\}_{T \in \mathcal{T}}$ such that $\{\lambda, \mathcal{T}\}$ is a convex combination for x for which $P(v, M, \Lambda)$ holds.*

Proof. As in the proof of Lemma 20, for each $st \in M$, pair the half-edges incident on s and pair those incident on t to obtain a collection of disjoint subsets of edges \mathcal{P} . Apply Theorem 17 to obtain $\{\lambda, \mathcal{T}\}$ which is a convex combination for x , where \mathcal{T} is a set of \mathcal{P} -rainbow v -trees (i.e., $x = \sum_{T \in \mathcal{T}} \lambda_T \chi^T$). Notice that this convex combination clearly satisfies the second requirement in Definition 19.

Now let $\delta(v) = \{e_v, f, g\}$, where w_1 and w_2 are the other endpoints of f and g , respectively. Without loss of generality, assume $e_{w_1} \notin M$. Since $x = \sum_{T \in \mathcal{T}} \lambda_T \chi^T$, we have $e_v \in T$ for $T \in \mathcal{T}$, since $x_{e_v} = 1$. In addition, we have $|\delta_T(v)| = 2$ for all $T \in \mathcal{T}$ by the definition of v -trees. Hence, $\sum_{T \in \mathcal{T}: f \in T, g \notin T} \lambda_T = \sum_{T \in \mathcal{T}: f \notin T, g \in T} \lambda_T = x_f = \frac{1}{2}$. Without loss of generality, assume $f \in T$ and $g \notin T$ for $T \in \mathcal{T}_f$, and $f \notin T$ and $g \in T$ for $T \in \mathcal{T}_g$, where $\mathcal{T}_f \cup \mathcal{T}_g = \mathcal{T}$ and $\mathcal{T}_f \cap \mathcal{T}_g = \emptyset$.

We can also assume that there are subsets $\mathcal{T}_f^1 \subseteq \mathcal{T}_f$ and $\mathcal{T}_g^1 \subseteq \mathcal{T}_g$ such that $\sum_{T \in \mathcal{T}_f^1} \lambda_T = \Lambda$ and $\sum_{T \in \mathcal{T}_g^1} \lambda_T = \Lambda$, since $\Lambda \leq \frac{1}{2}$. For $T \in \mathcal{T}_f^1$, replace T with $T - f$. Similarly, for $T \in \mathcal{T}_g^1$, replace T with $T + f$. For all $T \in \mathcal{T} \setminus (\mathcal{T}_f^1 \cup \mathcal{T}_g^1)$, keep T as is. Observe that $T \setminus \delta_T(v)$ still induces a spanning subgraph on $V \setminus \{v\}$ since we did not remove any edge in $T \setminus \delta(v)$ from the v -tree T . We want to show that the new convex combination $\{\lambda, \mathcal{T}\}$ is the desired convex combination for x . Notice that

$$\begin{aligned} \sum_{T \in \mathcal{T}} \lambda_T \chi_f^T &= \sum_{T \in \mathcal{T}_f^1} \lambda_T \chi_f^T + \sum_{T \in \mathcal{T}_f \setminus \mathcal{T}_f^1} \lambda_T \chi_f^T + \sum_{T \in \mathcal{T}_g^1} \lambda_T \chi_f^T + \sum_{T \in \mathcal{T}_g \setminus \mathcal{T}_g^1} \lambda_T \chi_f^T \\ &= 0 + \left(\frac{1}{2} - \Lambda\right) + \Lambda + 0 = x_f. \end{aligned}$$

So $x = \sum_{T \in \mathcal{T}} \lambda_T \chi^T$. Also, $T \in \mathcal{T}$ is a connected subgraph of G_x since each $T \in \mathcal{T}_f^1$ is obtained by removing an edge incident on v , which does not disconnect it. Finally, for each vertex s with $e_s \in M$, we have $|\delta_T(s)| = 2$ for all $T \in \mathcal{T}$. To observe this, notice that the initial convex combination satisfies this property for vertex s (since the convex combination is obtained via Theorem 17). In the transformation of the convex combination we only change edges incident on w_1 and w_2 , so if $s \neq w_1, w_2$ the property clearly still holds after the transformation. If $s = w_1$ or w_2 , we only remove or add an edge incident on s if $e_s \notin M$. ◀

2.2 Tools for parity correction

Let $G = (V, E)$ be an arbitrary graph and $O \subseteq V$ where $|O|$ is even. An O -join of G is a subgraph J of G in which the set of odd-degree vertices of J are exactly O . The convex hull of characteristic vectors of O -joins of G , denoted by O -JOIN(G) can be described as follows.

$$\begin{aligned} O\text{-JOIN}(G) &= \{x \in [0, 1]^E : \\ &\quad x(\delta(S) \setminus U) - x(U) \geq 1 - |U| \text{ for } S \subseteq V, U \subseteq \delta(S), |S \cap O| + |U| \text{ odd}\}. \end{aligned}$$

► **Lemma 22.** *Let x be a half-cycle point and assume that $G_x = (V, E_x)$ has no critical cuts. Let $M \subset W_x$ be a subset of 1-edges of G_x such that each 3-edge cut in G_x contains at most one edge from M . Let $O \subseteq V$ be a subset of vertices such that $|O|$ is even and for all $e = st \in M$, neither s nor t is in O . Also for any set $S \subseteq V$ such that $|\delta(S)| = 2$, both $|S \cap O|$ and $|\delta(S) \cap M|$ are even. Define vector z as follows: $z_e = \frac{1}{2}$ if $e \in W_x$ and $e \notin M$, and $z_e = \frac{1}{4}$ otherwise. Then vector $z \in O$ -JOIN(G_x).*

The proof of Lemma 22 can be found in the full version [12].

► **Observation 1.** *Let $G = (V, E)$ be a cubic graph, and let $O \subseteq V$ be a subset of vertices such that $|O|$ is even. Let $z \in O\text{-JOIN}(G)$, and $z(\delta(u)) \leq 1$ for all $u \in V$. Then there exists a set of O -joins of G , namely \mathcal{J} , and a probability distribution $\psi = \{\psi_J\}_{J \in \mathcal{J}}$ such that $\{\psi, \mathcal{J}\}$ is a convex combination for z . Moreover, for each vertex $v \in V$, the following properties hold.*

1. *If $u \in O$, then we have $|J \cap \delta(u)| = 1$ for each $J \in \mathcal{J}$. (Notice that in this case we must have $z(\delta(u)) = 1$.)*
2. *If $u \notin O$ and $\delta(u) = \{e, f, g\}$, then we have the following (four) cases. (Notice that sum of the right hand sides is exactly 1.)*

$$\sum_{J \in \mathcal{J}: J \cap \delta(u) = \emptyset} \psi_J = 1 - \frac{z(\delta(u))}{2},$$

$$\sum_{J \in \mathcal{J}: J \cap \delta(u) = \{h, h'\}} \psi_J = \frac{z(\delta(u))}{2} - z_{h''} \quad \text{for any distinct } h, h', h'' \in \delta(u).$$

The proof of this observation follows from the fact that if $z \in O\text{-JOIN}(G)$, then it can be efficiently decomposed into a convex combination of O -joins of G [10].

2.3 Convex combinations of tours: Base case

Let x be a half-cycle point such that $G_x = (V, E_x)$ has no critical cuts. Let v be a fixed vertex in V and let $\gamma(v) = \{w_1, w_2\}$. Let $\{M_1, \dots, M_h\}$ be a partition of W_x into induced matchings such that $|M_i \cap \{e_v, e_{w_1}, e_{w_2}\}| \leq 1$ for all $i \in [h]$, $e_v \in M_1$, each 3-edge cut of G_x contains at most one edge from each M_i , and each 2-edge cut of G_x contains an even number of edges from each M_i . Let $\alpha = \frac{1}{h}$ and Λ be some constant where $0 \leq \Lambda \leq \frac{1-\alpha}{2}$.

For $i = 1$, let \mathcal{T}_1 be a set of spanning subgraphs of G_x and let $\{\theta, \mathcal{T}_1\}$ be a convex combination for x for which $P(v, M_1, 0)$ holds (by Lemma 20). For $i \in \{2, \dots, \ell\}$, let \mathcal{T}_i be a set of spanning subgraphs of G_x and let $\{\theta, \mathcal{T}_i\}$ be a convex combination for x for which $P(v, M_i, \frac{\Lambda}{1-\alpha})$ holds (by Lemma 21). Notice that $\frac{\Lambda}{1-\alpha} \leq \frac{1}{2}$ since $\Lambda \leq \frac{1-\alpha}{2}$. Let $\mathcal{T} = \cup_{i \in [h]} \mathcal{T}_i$.

We can write x as a convex combination of the spanning subgraphs in \mathcal{T} , by weighting each set \mathcal{T}_i by α . In particular, we have $x = \alpha \sum_{i=1}^h \sum_{T \in \mathcal{T}_i} \theta_T \chi^T$. For each $T \in \mathcal{T}$, let $\sigma_T = \alpha \cdot \theta_T$. Then $\{\sigma, \mathcal{T}\}$ is a convex combination for x . From Definition 19 and Lemmas 20 and 21, we observe the following.

► **Claim 23.** For each $T \in \mathcal{T}$, $T \setminus \delta(v)$ induces a connected, spanning subgraph on $V \setminus \{v\}$.

For each $i \in [h]$, define $z_e^i = \frac{1}{2}$ if $e \in W_x \setminus M_i$ and $z_e^i = \frac{1}{4}$ otherwise. For each $T \in \mathcal{T}_i$, let $O_T \subseteq V$ be the set of odd-degree vertices of T . By construction, we have $V(M_i) \cap O_T = \emptyset$. By Lemma 22, we have $z^i \in O_T\text{-JOIN}(G)$, so there exists a set of O -joins \mathcal{J}_T and a probability distribution $\psi = \{\psi_J\}_{J \in \mathcal{J}_T}$ such that $\{\psi, \mathcal{J}_T\}$ is a convex combination for z^i . This implies that $x + z^i$ can be written as a convex combination of tours of G_x . We denote this set of tours by \mathcal{F}_i and we let $\mathcal{F} = \cup_{i \in [h]} \mathcal{F}_i$. We claim that $\sum_{i=1}^h \alpha(x + z^i)$ can be written as a convex combination of tours of G_x in \mathcal{F} using the probability distribution $\phi = \{\phi_F\}_{F \in \mathcal{F}}$, constructed as follows: For a tour F that is the union of $T \in \mathcal{T}$ and $J \in \mathcal{J}_T$, set $\phi_F = \sigma_T \cdot \psi_J$.

► **Claim 24.** Let x be a half-cycle point such that $G_x = (V, E_x)$ contains no critical cuts. Define vector $y \in \mathbb{R}^E$ as $y_e = \frac{3}{2} - \frac{\alpha}{4}$ for $e \in W_x$ and $y_e = \frac{3}{4}$ for $e \in H_x$. Then $\{\phi, \mathcal{F}\}$ is a convex combination for y .

Proof. We need to show that $y = \sum_{i=1}^h \alpha(x + z^i)$. First, let e be a 1-edge of G_x and M_j be the induced matching that contains e . Then, $x_e = 1$, $z_e^i = \frac{1}{2}$ for $i \in [h] \setminus \{j\}$ and $z_e^j = \frac{1}{4}$. Hence,

$$\sum_{i=1}^h \alpha(x_e + z_e^i) = \sum_{\ell=1}^h \alpha \cdot \frac{3}{2} - \alpha \cdot \frac{1}{4} = \frac{3}{2} - \frac{\alpha}{4}.$$

For a half-edge e of G_x , we have $x_e = \frac{1}{2}$ and $z_e^i = \frac{1}{4}$ for $i \in [h]$, so $\sum_{i=1}^h \alpha(x_e + z_e^i) = \frac{3}{4}$. \triangleleft

Now we prove some additional useful properties of the convex combination $\{\phi, \mathcal{F}\}$. For a vertex u such that $\delta(u) = \{e_u, f, g\}$ (i.e., where e_u is a 1-edge and f and g are half-edges), let \mathbb{P}_u denote the following set of patterns of edges such that u has even degree and the 1-edge e_u is included at least once.

$$\mathbb{P}_u = \{\{2e_u\}, \{e_u, f\}, \{e_u, g\}, \{2e_u, 2f\}, \{2e_u, 2g\}, \{2e_u, f, g\}, \{e_u, 2f, g\}, \{e_u, f, 2g\}\}.$$

Let $\mathbb{P} = \cup_{u \in V} \mathbb{P}_u$. For $0 \leq \alpha, \rho \leq 1$, define the function $\zeta_{\alpha, \rho} : \mathbb{P} \rightarrow [0, 1]$ as follows.

$$\zeta_{\alpha, \rho}(p_u) = \begin{cases} \frac{2-\alpha}{8} & \text{for } p_u = \{2e_u, f, g\}; \\ \frac{\rho}{2} & \text{for } p_u = \{2e_u\}; \\ \frac{\alpha+4\rho}{16} & \text{for } p_u \in \{\{e_u, 2f, g\}, \{e_u, f, 2g\}\}; \\ \frac{4+\alpha-4\rho}{16} & \text{for } p_u \in \{\{e_u, f\}, \{e_u, g\}\}; \\ \frac{2-\alpha-4\rho}{16} & \text{for } p_u \in \{\{2e_u, 2f\}, \{2e_u, 2g\}\}. \end{cases}$$

\triangleright **Claim 25.** The convex combination $\{\phi, \mathcal{F}\}$, has the following properties.

(i) For each vertex $u \in V$ there is a some constant η_u where $0 \leq \eta_u \leq \frac{1-\alpha}{2}$ and

$$\sum_{F \in \mathcal{F}: F \cap \delta(u) = p_u} \phi_F = \zeta_{\alpha, \eta_u}(p_u) \text{ for } p_u \in \mathbb{P}_u.$$

(ii) $\eta_v = \Lambda$.

The proof of Claim 25 can be found in the full version [12].

\blacktriangleright **Lemma 26.** Let x be a half-cycle point, and assume $G_x = (V, E_x)$ does not have any critical cuts. Let r be a vertex in V and let $\gamma(r) = \{w_1, w_2\}$. The set of 1-edges in G_x , W_x , can be partitioned into five induced matchings $\{M_1, \dots, M_5\}$ such that for $i \in \{1, \dots, 5\}$, the following properties hold.

(i) $M_i \cap \{e_r, e_{w_1}, e_{w_2}\} \leq 1$,

(ii) For $S \subseteq V$ such that $|\delta(S)| = 3$, $|\delta(S) \cap M_i| \leq 1$.

(iii) For $S \subseteq V$ such that $|\delta(S)| = 2$, $|\delta(S) \cap M_i|$ is even.

The proof of Lemma 26, which can be found in the full version, uses induction (i.e., gluing solutions for base cases) and the proof of the base case is an application of Brooks' theorem on a slight modification of the line graph of G_x [12].

Let $\gamma_v = \{w_1, w_2\}$. By Lemma 26, there are $\{M_1, \dots, M_5\}$ that partition W_x into induced matchings such that $|M_i \cap \{e_v, e_{w_1}, e_{w_2}\}| \leq 1$ for all $i \in [5]$, and each induced matching intersects a 3-edge-cut at most once and a 2-edge cut an even number of times. The following Lemma follows from Claim 25 by setting $\alpha = \frac{1}{5}$.

\blacktriangleright **Lemma 27.** Let x be a half-cycle point such that $G_x = (V, E_x)$ contains no critical cuts. Fix any vertex in $v \in V$ and Λ with $0 \leq \Lambda \leq \frac{2}{5}$. Define $y \in \mathbb{R}^E$ as $y_e = \frac{3}{2} - \frac{1}{20}$ for $e \in W_x$ and $y_e = \frac{3}{4}$ if $e \in H_x$. Then there is a set of tours of G_x denoted by \mathcal{F} and a probability distribution $\phi = \{\phi_F\}_{F \in \mathcal{F}}$ such that $\{\phi, \mathcal{F}\}$ is a convex combination for y . Moreover, this convex combination has the following properties.

(i) For each vertex $u \in V$, there is a some constant η_u where $0 \leq \eta_u \leq \frac{2}{5}$ and

$$\sum_{F \in \mathcal{F}: F \cap \delta(u) = p_u} \phi_F = \zeta_{\frac{1}{5}, \eta_u}(p_u) \text{ for } p_u \in \mathcal{P}_u.$$

(ii) $\eta_v = \Lambda$.

(iii) $F \setminus \delta_F(v)$ induces a connected multigraph on $V \setminus \{v\}$ for each $F \in \mathcal{F}$.

2.4 Convex combinations of tours: Gluing over critical cuts

► **Theorem 7.** Let $x \in \mathbb{R}_{\geq 0}^E$ be a half-cycle point. Define vector $y \in \mathbb{R}^E$ as follows: $y_e = \frac{3}{2} - \frac{1}{20}$ for $e \in W_x$ and $y_e = \frac{3}{4}$ for $e \in H_x$. Then $y \in \text{TSP}(G_x)$.

For a graph $G = (V, E)$ and nonempty subset of vertices $S \subset V$, contract the component induced on $\bar{S} = V \setminus S$ into a vertex and call this vertex $v_{\bar{S}}$. We define the graph G_S to be the graph induced on vertex set $S \cup v_{\bar{S}}$. The graph $G_{\bar{S}}$ is analogously defined on the vertex set $\bar{S} \cup v_S$.

► **Lemma 28.** Consider a graph $G = (V, E)$ and nonempty $S \subset V$ such that $\delta(S)$ is a minimum cardinality cut in $G = (V, E)$. Let F_S be a tour in G_S and let $F_{\bar{S}}$ be a tour in $G_{\bar{S}}$ such that $\chi_e^{F_S} = \chi_e^{F_{\bar{S}}}$ for $e \in \delta(S)$. Moreover, assume that $F_S \setminus \delta(v_{\bar{S}})$ induces a connected multigraph on S . Then the multiset of edges F defined as $\chi_e^F = \chi_e^{F_S}$ for $e \in E(G_S)$ and $\chi_e^F = \chi_e^{F_{\bar{S}}}$ for $e \in E(G_{\bar{S}})$ is a tour of G .

Proof. It is clear that F induces an Eulerian multigraph on G , but we need to ensure that F is connected. For example, the tour induced on $F_S \setminus \delta(v_{\bar{S}})$ might not be connected. However, since the subgraph of F_S induced on the vertex set S is connected, the tour F is connected: each vertex in \bar{S} is connected to some vertex in S . ◀

► **Lemma 29.** Let x be a half-cycle point such that $G_x = (V, E_x)$. Define $y \in \mathbb{R}^E$ as $y_e = \frac{3}{2} - \frac{1}{20}$ for $e \in W_x$ and $y_e = \frac{3}{4}$ if $e \in H_x$. Then there is a set of tours of G_x denoted by \mathcal{F} and a probability distribution $\phi = \{\phi_F\}_{F \in \mathcal{F}}$ such that $\{\phi, \mathcal{F}\}$ is a convex combination for y . Moreover, this convex combination has the following property.

For each vertex $u \in V$, there is a some constant η_u where $0 \leq \eta_u \leq \frac{2}{5}$ and

$$\sum_{F \in \mathcal{F}: F \cap \delta(u) = p_u} \phi_F = \zeta_{\frac{1}{5}, \eta_u}(p_u) \text{ for } p_u \in \mathcal{P}_u.$$

Proof. If G_x does not contain a critical cut, we apply Lemma 27. Otherwise, set $G := G_x$ and conduct the following procedure: Find a cut $S_1 \subset V(G)$ such that $G_1 = G_{S_1}$ contains no critical cuts. Then set $G := G_{\bar{S}_1}$ and find a cut $S_2 \subset V(G)$ such that $G_2 = G_{S_2}$ contains no critical cuts, etc.

At the end of this procedure, we have a series of graphs $\{G_1, \dots, G_k\}$ such that for each $j \in [k]$, G_j is the support graph of a half-cycle point and contains no critical cuts. Therefore, each G_j is a base case and we can find a convex combination of tours applying the procedure described in Section 2.3.

We glue the tours together in reverse order according to their index beginning with G_k and G_{k-1} . The graph G_{k-1} corresponds to G_S for some vertex set S of G , where G is the graph at the beginning of iteration $k-1$ of the above procedure. Note that $G_{\bar{S}}$ equals G_k and it has no critical cuts. Therefore, after invoking Lemma 27 to find a convex combination of tours for $G_{\bar{S}}$, we invoke Lemma 27 on G_S with $v = v_{\bar{S}}$ and $\Lambda = \eta_{v_{\bar{S}}}$ based on the convex combination of tours returned for $G_{\bar{S}}$. Now in the tours returned, the patterns on vertex $v_{\bar{S}}$ match those of v_S in the convex combination of tours previously found for $G_{\bar{S}}$.

After having glued together the tours from G_{k-1} and G_k in this manner, we glue the resulting tours with those in G_{k-2} , etc., until we have found a convex combination of tours for G_x . ◀

3 Discussion

In this paper, we presented an algorithm to save on 1-edges for a half-cycle point. To fully resolve half-integer TSP, we need to be able to save on half-edges. Towards this goal, we proposed a “base case” when there is no proper minimum cut (Theorem 13). It is not clear how to combine this with a gluing approach similar to the one for half-cycle points described in Section 2. Thus, we close with the following open problem.

► **Problem 30.** *Let x be a half-cycle point. vector $y \in \mathbb{R}^{E_x}$ as follows: $y_e = \frac{3}{2}$ for $e \in W_x$ and $y_e = \frac{3}{4} - \delta$ for $e \in H_x$. Show there exists a constant $\delta > 0$ such that $y \in \text{TSP}(G_x)$.*

References

- 1 Michel L. Balinski. Integer programming: Methods, uses, computations. *Management Science*, 12(3):253–313, 1965.
- 2 Sylvia Boyd and Robert Carr. Finding low cost TSP and 2-matching solutions using certain half-integer subtour vertices. *Discrete Optimization*, 8(4):525–539, 2011.
- 3 Sylvia Boyd and Philippe Legault. Toward a 6/5 Bound for the Minimum Cost 2-Edge Connected Spanning Subgraph. *SIAM Journal on Discrete Mathematics*, 31(1):632–644, 2017.
- 4 Sylvia Boyd and András Sebő. The Salesman’s Improved Tours for Fundamental Classes. In *Proceedings of 19th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 111–122, 2017.
- 5 Robert Carr and R. Ravi. A new bound for the 2-edge connected subgraph problem. In *Proceedings of 6th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 112–125, 1998.
- 6 Robert Carr and Santosh Vempala. On the Held-Karp relaxation for the asymmetric and symmetric traveling salesman problems. *Mathematical Programming*, 100(3):569–587, 2004.
- 7 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.
- 8 Gerard Cornuejols, Denis Naddef, and William Pulleyblank. The traveling salesman problem in graphs with 3-edge cutsets. *Journal of the ACM*, 32(2):383–410, 1985.
- 9 George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.
- 10 Jack Edmonds and Ellis L. Johnson. Matching, Euler tours and the Chinese postman. *Mathematical Programming*, 5(1):88–124, 1973.
- 11 Arash Haddadan and Alantha Newman. Efficient constructions of convex combinations for 2-edge-connected subgraphs on fundamental classes. *CoRR*, abs/1811.09906, 2018. [arXiv:1811.09906](#).
- 12 Arash Haddadan and Alantha Newman. Towards improving Christofides algorithm for half-integer TSP. *CoRR*, abs/1907.02120, 2019. [arXiv:1907.02120](#).
- 13 Arash Haddadan, Alantha Newman, and R. Ravi. Shorter tours and longer detours: Uniform covers and a bit beyond. *CoRR*, abs/1707.05387, 2017. [arXiv:1707.05387](#).
- 14 Philippe Legault. Towards new bounds for the 2-edge connected spanning subgraph problem. Master’s thesis, University of Ottawa, 2017.
- 15 Tobias Mömke and Ola Svensson. Removing and adding edges for the traveling salesman problem. *Journal of the ACM*, 63(1):2, 2016.

56:12 Towards Improving Christofides for Half-Integer TSP

- 16 Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 550–559, 2011.
- 17 Frans Schalekamp, David P. Williamson, and Anke van Zuylen. 2-matchings, the traveling salesman problem, and the subtour LP: A proof of the Boyd-Carr conjecture. *Mathematics of Operations Research*, 39(2):403–417, 2013.
- 18 András Sebő, Yohann Benchetrit, and Matej Stehlik. Problems about uniform covers, with tours and detours. *Mathematisches Forschungsinstitut Oberwolfach Report*, 51:2912—2915, 2014.
- 19 András Sebő and Jens Vygen. Shorter tours by nicer ears: $7/5$ -approximation for the graph-TSP, $3/2$ for the path version, and $4/3$ for two-edge-connected subgraphs. *Combinatorica*, 34(5):597–629, 2014.
- 20 David B. Shmoys and David P. Williamson. Analyzing the Held-Karp TSP bound: A monotonicity property with application. *Information Processing Letters*, 35(6):281–285, 1990.
- 21 Laurence A. Wolsey. *Heuristic analysis, linear programming and branch and bound*, pages 121–134. Springer Berlin Heidelberg, 1980.

Counting to Ten with Two Fingers: Compressed Counting with Spiking Neurons

Yael Hitron

Department of Computer Science and Applied Mathematics, Weizmann Institute of Science,
Rehovot 76100, Israel
yael.hitron@weizmann.ac.il

Merav Parter

Department of Computer Science and Applied Mathematics, Weizmann Institute of Science,
Rehovot 76100, Israel
merav.parter@weizmann.ac.il

Abstract

We consider the task of measuring time with probabilistic threshold gates implemented by bio-inspired spiking neurons. In the model of *spiking neural networks*, network evolves in discrete rounds, where in each round, neurons fire in pulses in response to a sufficiently high membrane potential. This potential is induced by spikes from neighboring neurons that fired in the previous round, which can have either an excitatory or inhibitory effect.

Discovering the underlying mechanisms by which the brain perceives the duration of time is one of the largest open enigmas in computational neuro-science. To gain a better algorithmic understanding onto these processes, we introduce the *neural timer* problem. In this problem, one is given a time parameter t , an input neuron x , and an output neuron y . It is then required to design a minimum sized neural network (measured by the number of auxiliary neurons) in which every spike from x in a given round i , makes the output y fire for the subsequent t consecutive rounds.

We first consider a deterministic implementation of a neural timer and show that $\Theta(\log t)$ (deterministic) threshold gates are both sufficient and necessary. This raised the question of whether randomness can be leveraged to reduce the number of neurons. We answer this question in the affirmative by considering neural timers with spiking neurons where the neuron y is required to fire for t consecutive rounds with probability at least $1 - \delta$, and should stop firing after at most $2t$ rounds with probability $1 - \delta$ for some input parameter $\delta \in (0, 1)$. Our key result is a construction of a neural timer with $O(\log \log 1/\delta)$ spiking neurons. Interestingly, this construction uses only *one* spiking neuron, while the remaining neurons can be deterministic threshold gates. We complement this construction with a matching lower bound of $\Omega(\min\{\log \log 1/\delta, \log t\})$ neurons. This provides the first separation between deterministic and randomized constructions in the setting of spiking neural networks.

Finally, we demonstrate the usefulness of compressed counting networks for *synchronizing* neural networks. In the spirit of distributed synchronizers [Awerbuch-Peleg, FOCS'90], we provide a general transformation (or simulation) that can take any synchronized network solution and simulate it in an asynchronous setting (where edges have arbitrary response latencies) while incurring a small overhead w.r.t the number of neurons and computation time.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases stochastic neural networks, approximate counting, synchronizer

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.57

Related Version A full version of the paper is available at <https://arxiv.org/abs/1902.10369>.

Funding *Merav Parter*: Supported in part by BSF-NSF grants.

Acknowledgements We are grateful to Cameron Musco, Renan Gross and Eylon Yogev for various useful discussions.



© Yael Hitron and Merav Parter;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 57; pp. 57:1–57:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Understanding the mechanisms by which brain experiences time is one of the major research objectives in neuroscience [26, 2, 9]. Humans measure time using a global clock based on standardized units of minutes, days and years. In contrast, the brain perceives time using specialized neural clocks that define their own time units. Living organisms have various other implementations of biological clocks, a notable example is the circadian clock that gets synchronized with the rhythms of a day.

In this paper we consider the algorithmic aspects of measuring *time* in a simple yet biologically plausible model of *stochastic spiking neural networks* (SNN) [23, 24], in which neurons fire in discrete pulses, in response to a sufficiently high membrane potential. This model is believed to capture the spiking behavior observed in real neural networks, and has recently received quite a lot of attention in the algorithmic community [18, 19, 20, 17, 15, 28, 6]. In contrast to the common approach in computational neuroscience and machine learning, the focus here is not on general computation ability or broad learning tasks, but rather on specific algorithmic implementation and analysis.

The SNN network is represented by a directed weighted graph $G = (V, A, W)$, with a special set of neurons $X \subset V$ called *inputs* that have no incoming edges, and a subset of *output* neurons¹ $Y \subset V$. The neurons in the network can be either deterministic threshold gates or probabilistic threshold gates. As observed in biological networks, and departing from many artificial network models, neurons are either strictly inhibitory (all outgoing edge weights are negative) or excitatory (all outgoing edge weights are positive). The network evolves in discrete, synchronous *rounds* as a Markov chain, where the firing probability of every neuron in round τ depends on the firing status of its neighbors in the preceding round $\tau - 1$. For probabilistic threshold gates this firing is modeled using a standard sigmoid function. Observe that an SNN network is in fact, a *distributed network*, every neuron responds to the firing spikes of its *neighbors*, while having no global information on the entire network.

Remark. In the setting of SNN, unlike classical distributed algorithms (e.g., LOCAL or CONGEST), the algorithm is fully specified by the *structure* of the network. That is, for a given network, its dynamic is fully determined by the model. Hence, the key complexity measure here is the size of the network measured by the number of auxiliary neurons². For certain problems, we also care for the tradeoff between the size and the computation time.

1.1 Measuring Time with Spiking Neural Networks

We consider the algorithmic challenges of measuring time using networks of threshold gates and probabilistic threshold gates. We introduce the *neural timer* problem defined as follows:

Given an input neuron x , an output neuron y , and a time parameter t , it is required to design a small neural network such that any firing of x in a given round invokes the firing of y for exactly the next t rounds.

In other words, it is required to design a succinct timer, activated by the firing of its input neuron, that alerts when exactly t rounds have passed.

¹ In contrast to the definition of *circuits*, we do allow output neurons to have outgoing edges and self loops. The requirement will be that the value of the output neurons converges over time to the desired solution.

² I.e., neurons that are not the input or the output neurons.

A trivial solution with t auxiliary neurons can be obtained by taking a directed chain of length t (Fig. 1): the head of the chain has an incoming edge from the input x , the output y has incoming edges from the input x , and all the other t neurons on the chain. All these neurons are simple *OR*-gates, they fire in round τ if at least one of their incoming neighbors fired in round $\tau - 1$. Starting with the firing of x in round 0, in each round i , exactly one neuron, namely the i^{th} neuron on the chain fires, which makes y keep on firing for exactly t rounds until the chain fades out. In this basic solution, the network spends one neuron that counts +1 and dies. It is noteworthy that the neurons in our model are very simple, they do not have any memory, and thus cannot keep track of the firing history. They can only base their firing decisions on the firing of their neighbors in the *previous* round.

With such a minimal model of computation, it is therefore intriguing to ask how to beat this linear dependency (of network size) in the time parameter t . Can we count to ten using only two (memory-less) neurons? We answer this question in the affirmative, and show that even with just simple deterministic threshold gates, we can measure time up to t rounds using only $O(\log t)$ neurons. It is easy to see that this bound is tight when using deterministic neurons (even when allowing some approximation). The reason is that $o(\log t)$ neurons encode strictly less than t distinct configurations, thus in a sequence of t rounds, there must be a configuration that re-occurs, hence locking the system into a state in which y fires forever.

► **Theorem 1 (Deterministic Timers).** *For every input time parameter $t \in \mathbb{N}_{>0}$, (1) there exists a deterministic neural timer network \mathcal{N} with $O(\log t)$ deterministic threshold gates, (2) any deterministic neural timer requires $\Omega(\log t)$ neurons.*

This timer can be easily adapted to the related problem of *counting*, where the network should output the number of spikes (by the input x) within a time window of t rounds.

Does Randomness Help in Time Estimation? Neural computation in general, and neural spike responses in particular, are inherently stochastic [16]. One of our broader scope agenda is to understand the power and limitations of randomness in neural networks. Does neural computation become *easier* or *harder* due to the stochastic behavior of the neurons?

We define a randomized version of the neural timer problem that allows some slackness both in the approximation of the time, as well as allowing a small error probability. For a given error probability $\delta \in (0, 1)$, the output y should fire for at least t rounds, and must stop firing after at most $2t$ rounds³ with probability at least $1 - \delta$. It turns out that this randomized variant leads to a considerably improved solution for $\delta = 2^{-O(t)}$:

► **Theorem 2 (Upper Bound for Randomized Timers).** *For every time parameter $t \in \mathbb{N}_{>0}$, and error probability $\delta \in (0, 1)$, there exists a probabilistic neural timer network \mathcal{N} with $O(\min\{\log \log 1/\delta, \log t\})$ deterministic threshold gates plus additional random spiking neuron.*

Our starting point is a simple network with $O(\log 1/\delta)$ neurons, each firing independently with probability $1 - 1/t$. The key observation for improving the size bound into $O(\log \log 1/\delta)$ is to use the *time axis*: we will use a *single* neuron to generate random samples over time, rather than having *many* random neurons generating these samples in a *single* round. The deterministic neural counter network with time parameter of $O(\log 1/\delta)$ is used as a building block in order to gather the firing statistics of a single spiking neuron. In light of the $\Omega(\log t)$

³ Taking $2t$ is arbitrary here, and any other constant would work as well.

lower bound for deterministic networks, we get the first separation between deterministic and randomized solutions for error probability $\delta = \omega(1/2^t)$. This shows that randomness can help, but up to a limit: Once the allowed error probability is exponentially small in t , the deterministic solution is the best possible. Perhaps surprisingly, we show that this behavior is tight:

► **Theorem 3 (Lower Bound for Randomized Timers).** *Any SNN network for the neural timer problem with time parameter t , and error $\delta \in (0, 1)$ must use $\Omega(\min\{\log \log 1/\delta, \log t\})$ neurons.*

Neural Counters. Spiking neurons are believed to encode information via their firing rates. This underlies the *rate coding* scheme [1, 30, 11] in which the spike-count of the neuron in a given span of time is interpreted as a *letter* in a larger alphabet. In a network of memory-less spiking neurons, it is not so clear how to implement this rate dependent behavior. How can a neuron convey a complicated message over time if its neighboring neurons remember only its recent spike? This challenge is formalized by the following neural counter problem: Given an input neuron x , a time parameter t , and $\Theta(\log t)$ output neurons represented by a vector \bar{y} , it is required to design a neural network such that the output vector \bar{y} holds the binary representation of the number of times that x fired in a sequence of t rounds. As we already mentioned this problem is very much related to the neural timer problem and can be solved using $O(\log t)$ neurons. Can we do better?

The problem of maintaining a *counter* using a small amount of space has received a lot of attention in the *dynamic streaming* community. The well-known Morris algorithm [27, 10] maintains an approximate counter for t counts using only $\log \log t$ bits. The high-level idea of this algorithm is to increase the counter with probability of $1/2^{C'}$ where C' is the current read of the counter. The counter then holds the exponent of the number of counts. By following ideas of [10], carefully adapted to the neural setting, we show:

► **Theorem 4 (Approximate Counting).** *For every time parameter t , and $\delta \in (0, 1)$, there exists a randomized construction of approximate counting network using $O(\log \log t + \log(1/\delta))$ deterministic threshold gates plus an additional single random spiking neuron, that computes an $O(1)$ (multiplicative) approximation for the number of input spikes in t rounds with probability $1 - \delta$.*

We note that unlike the deterministic construction of timers that could be easily adopted to the problem of neural counting, our optimized randomized timers with $O(\log \log 1/\delta)$ neurons cannot be adopted into an approximate counter network. We therefore solve the latter by adopting Morris algorithm to the neural setting.

Broader Scope: Lessons From Dynamic Streaming Algorithms. We believe that approximate counting problem provides just one indication for the potential relation between succinct neural networks and dynamic streaming algorithms. In both settings, the goal is to gather statistics (e.g., over time) using a small amount of space. In the setting of neural network there are additional difficulties that do not show up in the streaming setting. E.g., it is also required to obtain fast *update time*, as illustrated in our solution to the approximate counting problem.

1.2 Neural Synchronizers

The standard model of spiking neural networks assumes that all edges (synapses) in the network have a uniform response latency. That is, the electrical signal is passed from the presynaptic neuron to the postsynaptic neuron within a fixed time unit which we call a

round. However, in real biological networks, the response latency of synapses can vary considerably depending on the biological properties of the synapse, as well as on the distance between the neighboring neurons. This results in an asynchronous setting in which different edges have distinct response time. We formalize a simple model of spiking neurons in the asynchronous setting, in which the given neural network also specifies a *response latency* function $\ell : A \rightarrow \mathbb{R}_{\geq 1}$ that determines the number of rounds it takes for the signal to propagate over the edge. Inspired by the synchronizers of Awerbuch and Peleg [4], and using the above mentioned compressed timer and counter modules, we present a general simulation methodology (a.k.a synchronizers) that takes a network $\mathcal{N}_{\text{sync}}$ that solves the problem in the synchronized setting, and transform it into an “analogous” network $\mathcal{N}_{\text{async}}$ that solves the same problem in the asynchronous setting.

The basic building blocks of this transformation is the neural time component adapted to the asynchronous setting. The cost of the transformation is measured by the overhead in the number of neurons and in the computation time. Using our neural timers leads to a small overhead in the number of neurons.

► **Theorem 5** (Synchronizer, Informal). *There exists a synchronizer that given a network $\mathcal{N}_{\text{sync}}$ with n neurons and maximum response latency⁴ L , constructs a network $\mathcal{N}_{\text{async}}$ that has an “analogous” execution in the asynchronous setting with a total number of $O(n + L \log L)$ neurons and a time overhead of $O(L^3)$.*

We note that although the construction is inspired by the work of Awerbuch and Peleg [4], due to the large differences between these models, the precise formulation and implementation of our synchronizers are quite different. The most notable difference between the distributed and neural setting is the issue of memory: in the distributed setting, nodes can aggregate the incoming messages and respond when all required messages have arrived. In strike contrast, our neurons can only respond (by either firing or not firing) to signals arrived in the *previous* round, and all signals from previous rounds cannot be locally stored. For this reason and unlike [4], we must assume a bound on the largest edge latency. In particular, in the full version we show that the size overhead of the transformed network $\mathcal{N}_{\text{async}}$ must depend, at least logarithmically, on the value of the largest latency L .

► **Observation 1.** *The size overhead of any synchronization scheme is $\Omega(\log L)$.*

This provably illustrates the difference in the overhead of synchronization between general distributed networks and neural networks. We leave the problem of tightening this lower bound (or upper bound) as an interesting open problem.

Additional Related Work. To the best of our knowledge, there are two main previous theoretical work on asynchronous neural networks. Maass [22] considered a quite elaborated model for deterministic neural networks with *arbitrary* response *functions* for the edges, along with latencies that can be chosen by the network designer. Within this generalized framework, he presented a coarse description of a synchronization scheme that consists of various time modules (e.g., initiation and delay modules). Our work complements the scheme of [22] in the simplified SNN model by providing a rigorous implementation and analysis for size and time overhead. Khun et al. [14] analyzed the synchronous and asynchronous behavior under the stochastic neural network model of DeVille and Peskin [7]. Their model and framework is quite different from ours, and does not aim at building synchronizers.

⁴ I.e., L correspond to the *length* of the longest round.

Turning to the setting of logical circuits, there is a long line of work on the asynchronous setting under various model assumptions [3, 12, 29, 5, 25] that do not quite fit the memory-less setting of spiking neurons.

Comparison with Concurrent Work [31]. Independently to our work, Wang and Lynch proposed a similar construction for the neural counter problem. Their work restricts attention to deterministic threshold gates and do not consider the neural timer problem and synchronizers which constitute the main contribution of our paper. We note that our approximate counter solution with $O(\log \log t + \log(1/\delta))$ neurons resolves the open problem stated in [31].

1.3 Preliminaries

We start by defining our model along with useful notation.

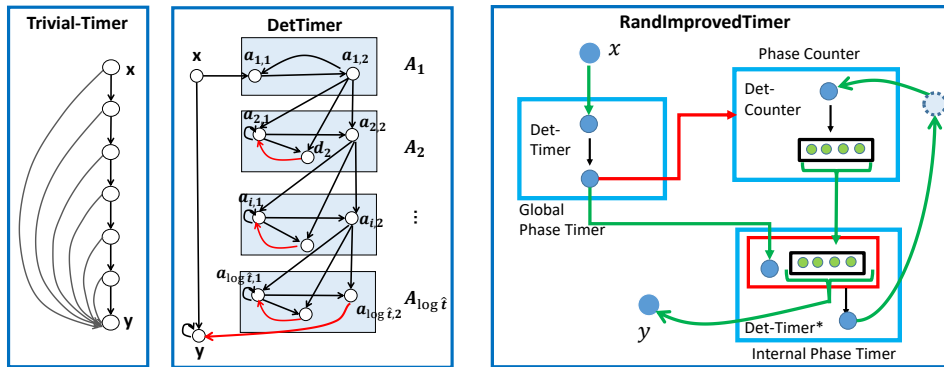
A Neuron. A *deterministic* neuron u is modeled by a *deterministic* threshold gate. Letting $b(u)$ to be the threshold value of u . Then it outputs 1 if the weighted sum of its incoming neighbors exceeds $b(u)$. A *spiking neuron* is modeled by a probabilistic threshold gate that fires with a sigmoidal probability $p(x) = \frac{1}{1+e^{-x}}$ where x is the difference between the weighted incoming sum of u and its threshold $b(u)$.

Neural Network Definition. A *Neural Network* (NN) $\mathcal{N} = \langle X, Z, Y, w, b \rangle$ consists of n input neurons $X = \{x_1, \dots, x_n\}$, m output neurons $Y = \{y_1, \dots, y_m\}$, and ℓ auxiliary neurons $Z = \{z_1, \dots, z_\ell\}$. In a *deterministic* neural network (DNN) all neurons are deterministic threshold gates. In spiking neural network (SNN), the neurons can be either deterministic threshold gates or probabilistic threshold gates. The directed weighted synaptic connections between $V = X \cup Z \cup Y$ are described by the weight function $w : V \times V \rightarrow \mathbb{R}$. A weight $w(u, v) = 0$ indicates that a connection is not present between neurons u and v . Finally, for any neuron v , $b(v) \in \mathbb{R}_{\geq 0}$ is the threshold value (activation bias). The weight function defining the synapses is restricted in two ways. The in-degree of every input neuron x_i is zero, i.e., $w(u, x) = 0$ for all $u \in V$ and $x \in X$. Additionally, each neuron is either inhibitory or excitatory: if v is inhibitory, then $w(v, u) \leq 0$ for every u , and if v is excitatory, then $w(v, u) \geq 0$ for every u .

Network Dynamics. The network evolves in discrete, synchronous rounds as a Markov chain. The firing probability of every neuron in round τ depends on the firing status of its neighbors in round $\tau - 1$, via a standard sigmoid function, with details given below. For each neuron u , and each round $\tau \geq 0$, let $u^\tau = 1$ if u fires (i.e., generates a spike) in round τ . Let u^0 denote the initial firing state of the neuron. The firing state of each input neuron x_j in each round is the input to the network. For each non-input neuron u and every round $\tau \geq 1$, let $pot(u, \tau)$ denote the membrane potential at round τ and $p(u, \tau)$ denote the firing probability ($\Pr[u^\tau = 1]$), calculated as:

$$pot(u, \tau) = \sum_{v \in V} w_{v,u} \cdot v^{\tau-1} - b(u) \text{ and } p(u, \tau) = \frac{1}{1 + e^{-\frac{pot(u, \tau)}{\lambda}}} \quad (1)$$

where $\lambda > 0$ is a *temperature parameter* which determines the steepness of the sigmoid. Clearly, λ does not affect the computational power of the network (due to scaling of edge weights and thresholds), thus we set $\lambda = 1$. In *deterministic* neural networks (DNN), each neuron u is a deterministic threshold gate that fires in round τ iff $pot(u, \tau) \geq 0$.



■ **Figure 1** Illustration of timer networks with time parameter t . Left: The naïve timer with $\Theta(t)$ neurons. Mid: deterministic timer with $\Theta(\log t)$ neurons. Right: randomized timer with $O(\log \log 1/\delta)$ neurons, using the DetTimer modules with parameter $t' = \log 1/\delta$.

Network States (Configurations). Given a network \mathcal{N} (either a DNN or SNN) with N neurons, the configuration (or *state*) of the network in time τ denoted as s_τ can be described as an N -length binary vector indicating which neuron fired in round τ .

The Memoryless Property. The neural networks have a memoryless property, in the sense that each state depends only on the state of the previous round. In a DNN network, the state $s_{\tau-1}$ fully determines s_τ . In an SNN network, for every fixed state s^* it holds $\Pr[s_\tau = s^* \mid s_1, \dots, s_{\tau-1}] = \Pr[s_\tau = s^* \mid s_{\tau-1}]$. Moreover for any $\tau, \tau', r > 0$, it holds that $\Pr[s_{\tau+r} = s^* \mid s_\tau] = \Pr[s_{\tau'+r} = s^* \mid s_{\tau'}]$.

Hard-Wired Inputs. We consider neural networks that *solve* a given parametrized problem (e.g., neural timer with time parameter t). The parameter to the problem can be either hard-wired in the network or alternatively be given as part of the input layer to the network. In most of our constructions, the time parameter is hard-wired. In some cases, we also show constructions with soft-wiring.

2 Deterministic Constructions of Neural Timer Networks

As a warm-up, we start by considering deterministic neural timers.

► **Definition 6** (Det. Neural Timer Network). *Given time parameter t , a deterministic neural timer network \mathcal{DT} is a network of threshold gates, with an input neuron x , an output neuron y , and additional auxiliary neurons. The network satisfies that in every round τ , $y^\tau = 1$ iff there exists a round $\tau > \tau' \geq \tau - t$ such that $x^{\tau'} = 1$.*

Lower Bound (Pf. of Thm. 1(2)). For a given neural timer network \mathcal{N} with N auxiliary neurons, recall that the *state* of the network in round τ is described by an N -length vector indicating the firing neurons in that round. Assume towards contradiction that there exists a neural timer with $N \leq \log t - 1$ auxiliary neurons. Since there are at most 2^N different states, by the pigeonhole principle, there must be at least two rounds $\tau, \tau' \leq t - 1$ in which the state of the network is identical, i.e., where $s_\tau = s_{\tau'} = s^*$ for some $s^* \in \{0, 1\}^N$. By the correctness of the network, the output neuron y fires in all rounds $\tau'' \in [\tau + 1, \tau' + 1]$. By the memoryless property, we get that $s_{\tau''} = s^*$ for $\tau'' = \tau + i \cdot (\tau' - \tau)$ for every $i \in \mathbb{N}_{\geq 0}$.

Thus y continues firing forever, in contradiction that it stops firing after t rounds. Note that this lower bound holds even if y is allowed to stop firing in any finite time window.

A Matching Upper Bound (Pf. Thm. 1(1)). For ease of explanation, we will sketch here the description of the network assuming that it is applied only once (i.e., the input x fires once within a window of t rounds). Taking care of the general case requires slight adaptations⁵, see the full version for complete details.

At the high-level, the network consists of $k = \Theta(\log t)$ layers A_1, \dots, A_k each containing two excitatory neurons $a_{i,1}, a_{i,2}$ denoted as *counting* neurons, and one inhibitory neuron d_i . Each layer A_i gets its input from layer A_{i-1} for every $i \geq 2$, and A_1 gets its input from x . The role of each layer A_i is to count *two* firing events of the neuron $a_{i-1,2} \in A_{i-1}$. Thus the neuron $a_{\log t, 2}$ counts $2^{\log t}$ rounds.

Because our network has an update time of $\log t$ rounds (i.e., number of rounds to update the timer), for a given time parameter t , the construction is based on the parameter \hat{t} where $\hat{t} + \log \hat{t} = t$. We assume that \hat{t} is a power of 2, the general case can also be solved with $O(\log t)$ neurons, the analysis is deferred to the full version.

- The first layer A_1 consists of two neurons $a_{1,1}, a_{1,2}$. The first neuron $a_{1,1}$ has positive incoming edges from x and $a_{1,2}$ with weights $w(x, a_{1,1}) = 3$, $w(a_{1,2}, a_{1,1}) = 1$, and threshold $b(a_{1,1}) = 1$. The second neuron $a_{1,2}$ has an incoming edge from $a_{1,1}$ with weight $w(a_{1,1}, a_{1,2}) = 1$ and threshold $b(a_{1,2}) = 1$. Because we have a loop going from $a_{1,1}$ to $a_{1,2}$ and back, once x fired $a_{1,2}$ will fire every two rounds.
- For every $i = 2 \dots \log \hat{t}$, the i^{th} layer A_i contains 3 neurons, two *counting* neurons $a_{i,1}, a_{i,2}$ and a reset neuron d_i . The first neuron $a_{i,1}$ has positive incoming edges from $a_{i-1,2}$, and a self loop with weight $w(a_{i-1,2}, a_{i,1}) = w(a_{i,1}, a_{i,1}) = 1$, a negative incoming edge from d_i with weight $w(d_i, a_{i,1}) = -1$, and threshold $b(a_{i,1}) = 1$. The second counting neuron $a_{i,2}$ has incoming edges from $a_{i-1,2}$ and $a_{i,1}$ with weight $w(a_{i-1,2}, a_{i,2}) = w(a_{i,1}, a_{i,2}) = 1$, and threshold $b(a_{i,2}) = 2$. The reset neuron d_i is an inhibitor copy of $a_{i-1,2}$ and therefore also has incoming edges from $a_{i-1,2}$ and $a_{i,1}$ with weight $w(a_{i-1,2}, d_i) = w(a_{i,1}, d_i) = 1$ and threshold $b(d_i) = 2$. As a result, $a_{i,1}$ starts firing after $a_{i-1,2}$ fires once, and $a_{i,2}$ fires after $a_{i-1,2}$ fires twice. Then the neuron d_i inhibits $a_{i,1}$ and the layer is ready for a new count.
- The output neuron y has a positive incoming edge from x as well as a self-loop with weights $w(x, y) = 2$, $w(y, y) = 1$. In addition, it has a negative incoming edge from the last counting neuron $a_{\log \hat{t}, 2}$ with weight $w(a_{\log \hat{t}, 2}, y) = -1$ and threshold $b(y) = 1$. Hence, after x fires the output y continues to fire as long as $a_{\log \hat{t}, 2}$ did not fire.
- The last counting neuron $a_{\log \hat{t}, 2}$ also have negative outgoing edges to all counting neurons (neurons of the form $a_{i,j}$) with weight $w(a_{\log \hat{t}, 2}, a_{i,j}) = -2$. As a result, after the timer counts t rounds it is reset.

Timer with Time Parameter. In the full version we show a slight modified variant of neural timer denoted by DetTimer^* which also receives as input an additional set of $\log t$ neurons that encode the desired duration of the timer. This modified variant is used in our improved randomized constructions.

⁵ I.e., whenever x fires again in a window of t rounds, one should reset the timer and start counting t rounds from that point on.

Neural Counters. We also show a modification of the timer into a counter network DetCounter that instead of counting the number of rounds, counts the number of input spikes in a time interval of t rounds. For complete details we defer the reader to the full version.

► **Lemma 7.** *Given time parameter t , there exists a deterministic neural counter network which has an input neuron x , a collection of $\log t$ output neurons represented by a vector \bar{y} , and $O(\log t)$ additional auxiliary neurons. In a time window of t rounds, for every round τ , if x fired r_τ times in the last τ rounds, the output \bar{y} encodes r_τ by round $\tau + \log r_\tau + 1$.*

This extra-additive factor of $\log r_\tau$ is due to the update time of the counter. In addition, in the full version we revisit the neural counter problem and provide an *approximate* randomized solution with $O(\log \log t + \log(1/\delta))$ many neurons where δ is the error parameter. This construction is based on the well-known Morris algorithm (using the analysis of [10]) for approximate counting in the streaming model.

3 Randomized Constructions of Neural Timer Networks

We now turn to consider randomized implementations. The input to the construction is a time parameter t and an error probability $\delta \in (0, 1)$, that are hard-wired into the network.

► **Definition 8** (Rand. Neural Timer Network). *A randomized neural timer \mathcal{RT} for parameters $t \in \mathbb{N}_{>0}$ and $\delta \in (0, 1)$, satisfies the following for a time window of $\text{poly}(t)$ rounds.*

- *For every fixed firing event of x in round τ , with probability $1 - \delta$, y fires in each of the following t rounds.*
- *$y^{\tau'} = 0$ for every round τ' such that $\tau' - \text{Last}(\tau') \geq 2t$ with probability $1 - \delta$, where $\text{Last}(\tau') = \max\{i \leq \tau' \mid x^i = 1\}$ is the last round τ in which x fired up to round τ' .*

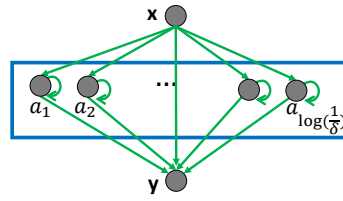
Note that in our definition, we have a success guarantee of $1 - \delta$ for any fixed firing event of x , on the event that y fires for t many rounds after this firing. In contrast, with probability of $1 - \delta$ over the entire span of $\text{poly}(t)$ rounds, y does not fire in cases where the last firing of x was $2t$ rounds apart. We start by showing a simple construction with $O(\log 1/\delta)$ neurons.

3.1 Warm Up: Randomized Timer with $O(\log 1/\delta)$ Neurons

The network BasicRandTimer(t, δ) contains a collection of $\ell = \Theta(\log 1/\delta)$ spiking neurons $A = \{a_1, \dots, a_\ell\}$ that can be viewed as a *time-estimator population*. Each of these neurons have a positive self loop, a positive incoming edge from the input neuron x , and a positive outgoing edge to the output neuron y . See Figure 2 for an illustration. Whereas these a_i neurons are probabilistic spiking neurons⁶, the output y is simply a threshold gate. We next explain the underlying intuition. Assume that the input x fired in round 0. It is then required for the output neuron y to fire for at least t rounds $1, \dots, t$, and stop firing after at most $2t$ rounds with probability $1 - \delta$. By having every neuron a_i firing (independently) w.p $(1 - 1/t)$ in each round given that it fired in the previous round⁷, we get that a_i fires for t consecutive rounds with probability $(1 - 1/t)^t = 1/e$. On the other hand, it fires for $2t$ consecutive rounds with probability $(1 - 1/t)^{2t} = 1/e^2$. Since we have $\Theta(\log 1/\delta)$ many neurons, by a simple application of Chernoff bound, the output neuron y (which simply counts the number of firing neurons in A) can distinguish between round t and round $2t$ with probability $1 - \delta$, see the full version for the complete proof.

⁶ A neuron that fires with a probability specified in Eq. (1)

⁷ A neuron a_i that stops firing in a given round, drops out and would not fire again with good probability.



■ **Figure 2** Illustration of the $\text{BasicRandTimer}(t, \delta)$ network. Each neuron a_i fires with probability $1 - 1/t$ in round τ given that it fired in the previous round $\tau - 1$, and therefore fires for t consecutive rounds with constant probability. The output y fires if at least $1/(2e)$ fraction of the a_i neurons fired in the previous round.

3.2 Improved Construction with $O(\log \log 1/\delta)$ Neurons

We next describe an optimal randomized timer RandImprovedTimer with an exponentially improved number of auxiliary neurons. This construction also enjoys the fact that it requires a *single* spiking neuron, while the remaining neurons can be deterministic threshold gates. Due to the tightness of Chernoff bound, one cannot really hope to estimate time with probability $1 - \delta$ using $o(\log(1/\delta))$ samples. Our key idea here is to generate the same number of samples by re-sampling one particular neuron over several rounds. Intuitively, we are going to show that for our purposes having $\ell = \log(1/\delta)$ neurons a_1, \dots, a_ℓ firing with probability $1 - 1/t$ in a *given* round is *equivalent* to having a *single* neuron a^* firing with probability $1 - 1/t$ (independently) in a sequence of ℓ rounds.

Specifically, observe that the distinction between round t and $2t$ in the BasicRandTimer network is based only on the *number* of spiking neurons in a given round. In addition, the distribution on the number of times a^* fires in a span of ℓ rounds is equivalent to the distribution on the number of firing neurons a_1, \dots, a_ℓ in a given round. For this reason, every phase of RandImprovedTimer simulates a single round of BasicRandTimer . To count the number of firing events in ℓ rounds, we use the deterministic neural counter module with $\log \ell = O(\log \log 1/\delta)$ neurons.

We now further formalize this intuition. The network RandImprovedTimer simulates each round of BasicRandTimer using a phase of $\ell' = \Theta(\log 1/\delta)$ rounds⁸, but with only $O(\log \log 1/\delta)$ neurons. In the BasicRandTimer network each of the neurons a_i fires (independently) in each round w.p $1 - 1/t$. Once it stops firing in a given round, it basically drops out and would not fire again with good probability. Formally, consider an execution of the BasicRandTimer and let n_i be the number of neurons in A that fired in round i . In round $i + 1$ of this execution, we have n_i many neurons each firing w.p $1 - 1/t$ (while the remaining neurons in A fire with a very small probability). In the corresponding $i + 1$ phase of the network RandImprovedTimer , the chief neuron a^* fires w.p $1 - 1/t'$ where $t' = \frac{t}{\ell'}$ for $n'_i \leq \ell$ consecutive rounds⁹ where n'_i is the number of rounds in which a^* fired in phase i .

The dynamics of the network RandImprovedTimer is based on discrete phases. Each phase has a fixed number of $\ell' = O(\ell)$ rounds, but has a possibly different number of *active rounds*, namely, rounds in which a^* attempts firing. Specifically, a phase i has an active part of n'_i rounds where n'_i is the number of rounds in which a^* fired in phase $i - 1$. In the remaining $\ell' - n'_i$ rounds of that phase, a^* is idle. To implement this behavior, the network should keep

⁸ Due to tactical reasons each phase consists of $\ell' = \ell + \log \ell$ rounds instead of ℓ .

⁹ Note that because each phase takes $\ell' = \Theta(\log 1/\delta)$ rounds, we will need to count $t' = \frac{t}{\ell'}$ many phases. Thus a^* fires with probability $1 - 1/t'$ rather than w.p $1 - 1/t$.

track of the number of rounds in which a^* fires in each phase, and supply it as an input to the next phase (as it determines the length of the active part of that phase). For that purpose we will use the deterministic modules of neural timers and counters. The module `DetCounter` with time parameter $\Theta(\log 1/\delta)$ is responsible for counting the number of rounds that a^* fires in a given phase i . The output of this module at the end of the phase is the input to a `DetTimer*` module¹⁰ in the beginning of phase $i + 1$. In addition, we also need a *phase timer* module `DetTimer` with time parameter $\Theta(\log 1/\delta)$ that “announces” the end of a phase and the beginning of a new one. Similarly to the network `BasicRandTimer`, the output neuron y fires as long as a^* fires for at least $(1/2e)$ fraction of the rounds in each phase (in an analogous manner as in the `BasicRandTimer` construction). See Fig. 1 for an illustration of the network. Note that since we only use deterministic modules with time parameter $\Theta(\log 1/\delta)$, the total number of neurons (which are all threshold gates) will be bounded by $O(\log \log 1/\delta)$. See the full version for the complete proof of Thm. 2.

Remark. We note that the fact that one can get a randomized upper bound of $O(\log \log 1/\delta)$ neurons has to do with the fact that our spiking neurons can be set to fire with probability $1 - 1/t$. Therefore the value of t is hard-wired in the network. We also note that in a more restrictive setting where neurons are simple fair coins that fire with probability half in each round, the size complexity might be dependent in t .

3.3 A Matching Lower Bound

We now turn to provide a proof sketch for Thm. 2. The full proof can be found in the full version. Assume towards contradiction that there exists a randomized neural timer \mathcal{N} for a given time parameter t with $N = o(\log \log 1/\delta)$ neurons that succeeds with probability at least $1 - \delta$. This implies that there exists some constant $c \geq 2$ such that y stops firing after $(c - 1) \cdot t$ rounds w.p $1 - \delta$. Since we have N many neurons, the number of distinct states (or configurations) is bounded by $S = 2^N = o(\log 1/\delta)$. In particular, we will adjust the constant in the number of neurons N such that $S \leq \frac{\log 1/\delta}{\log \log 1/\delta}$. Since $t > \log(1/\delta)$ ¹¹, in every execution of \mathcal{N} for at least t rounds, there must be a state that occurs at least *twice* during the execution. Moreover, the sequence of t rounds consists of at least $2S$ disjoint intervals each of length $t/3S$. We therefore get that in every execution of the network, there must be a state that occurs at least twice for some rounds t', t'' such that $t'' - t' > t/3S$. In other words, we have the guarantee that there is always some state that reoccurs after a sufficiently large number of rounds. Since there are at most S configurations, we conclude that there must be one particular configuration s^* for which the probability to reoccur (after at least $t/(3S)$ rounds) is at least $1/S = \Omega(1/\log(1/\delta))$.

Let Π be the family of all $(c \cdot t)$ -length executions of \mathcal{N} . We now restrict attention to all those executions in which s^* reoccurs within the first t rounds, with spacing of $\Omega(t/\log(1/\delta))$ rounds between its appearances¹². We call the subset $\Pi^* \subseteq \Pi$ of those executions *special*. In addition, an execution in Π is *good* if y fires in *each* of the first t rounds, otherwise it is *bad*.

We next claim that the probability of an execution to be both special and good is at least $p^* = \Omega(1/\log(1/\delta) - \delta)$. First, by the definition of Π^* , the probability of an execution to be special is at least $1/S = \Omega(1/\log(1/\delta))$. In addition, by the success guarantee of \mathcal{N} , y fires in the first t rounds w.p. at least $1 - \delta$. Thus by the union bound, we get that the probability of a special and good execution is $1/S - \delta$. This allows us to also conclude that given that s^*

¹⁰ Here we use the variant of `DetTimer` in which the time is encoded in the input layer of the network.

¹¹ The lower bound is meaningful only when $\delta = 2^{-O(t)}$.

¹² We do allow s^* to appear several times within this interval.

appears in some round $t' < t - t/3S$, with probability of at least p^* the following happens: (1) s^* reoccurs in some round t'' such that $t'' - t \in [t/3S, t]$ and (2) y fires during the entire interval $[t', t'']$. We now use this argument to conclude that w.p. at least δ , the output y fires for at least $c \cdot t$ rounds, which will lead to a contradiction. To see this claim, note that w.p. at least $1/S$ there is a round $t' < t - t/3S$ in which s^* appears. With probability at least p^* there is a round $t'' > t + t/3S$ in which s^* appears again and y fires in each of the rounds in $[t', t'']$. A time window of $c \cdot t$ rounds contains at most $3c \cdot S$ intervals of length $t/3S$. Thus by the memory-less property, we get that the probability s^* reoccurs every $[t/3S, t]$ rounds and that y fires after ct rounds is at least $1/S \cdot (p^*)^{3c \cdot S} > \delta$, contradiction as y fires after $c \cdot t$ rounds w.p at most δ .

4 Applications to Synchronizers

The Asynchronous Setting. In this setting, the neural network $\mathcal{N} = \langle X, Z, Y, w, b \rangle$ also specifies a response latency function $\ell : A \rightarrow \mathbb{N}_{>0}$. For ease of notation, we normalize all latency values such that $\min_{e \in A} \ell(e) = 1$ and denote the maximum response latency by $L = \max_{e \in A} \ell(e)$. Supported by biological evidence [13], we assume that self-loop edges (a.k.a. autapses) have the minimal latency in the network, that $\ell((u, u)) = 1$ for self-edges (u, u) . This assumption is crucial in our design¹³. Indeed the exceptional short latency of self-loop edges has been shown to play a critical role in biological network synchronization [21, 8]. The dynamics proceeds in synchronous rounds and phases. The length of a round corresponds to the minimum edge latency, this is why we normalize the latency values so that $\min_{e \in A} \ell(e) = 1$. If neuron u fires in round τ , its endpoint v receives u 's signal in round $\tau + \ell(e)$. Formally, a neuron u fires in round τ with probability $p(u, \tau)$:

$$pot(u, \tau) = \sum_{v \in X \cup Z \cup Y} w_{v,u} \cdot v^{\tau - \ell(u,v)} - b(u) \text{ and } p(u, \tau) = \frac{1}{1 + e^{-\frac{pot(u,\tau)}{\lambda}}} \quad (2)$$

Synchronizer. A synchronizer ν is an algorithm that gets as input a network $\mathcal{N}_{\text{sync}}$ and outputs a network $\mathcal{N}_{\text{async}} = \nu(\mathcal{N}_{\text{sync}})$ such that $V(\mathcal{N}_{\text{sync}}) \subseteq V(\mathcal{N}_{\text{async}})$ where $V(\mathcal{N})$ denotes the neurons of a network \mathcal{N} . The network $\mathcal{N}_{\text{async}}$ works in the asynchronous setting and should have *similar execution* to $\mathcal{N}_{\text{sync}}$ in the sense that for every neuron $v \in V(\mathcal{N}_{\text{sync}})$, the firing pattern of v in the asynchronous network should be similar to the one in the synchronous network. The output network $\mathcal{N}_{\text{async}}$ simulates each round of the network $\mathcal{N}_{\text{sync}}$ as a *phase*.

► **Definition 9 (Pulse Generator and Phases).** A pulse generator is a module that fires to declare the end of each phase. Denote by $t(v, p)$ the (global) round in which neuron v receives the p^{th} spike from the pulse generator. We say that v is in phase p during all rounds $\tau \in [t(v, p - 1), t(v, p)]$.

► **Definition 10 (Similar Execution (Deterministic Networks)).** The synchronous execution Π_{sync} of a deterministic network $\mathcal{N}_{\text{sync}}$ is specified by a list of states $\Pi_{\text{sync}} = \{\sigma_1, \dots, \}$ where each σ_i is a binary vector describing the firing status of the neurons in round i . The asynchronous execution of network $\mathcal{N}_{\text{async}}$ denoted by Π_{async} is defined analogously only when applying the asynchronous dynamics (of Eq. (2)). The execution Π_{async} is divided into phases of fixed length. The networks $\mathcal{N}_{\text{sync}}$ and $\mathcal{N}_{\text{async}}$ have a similar execution if $V(\mathcal{N}_{\text{sync}}) \subseteq V(\mathcal{N}_{\text{async}})$, and in addition, a neuron $v \in V(\mathcal{N}_{\text{sync}})$ fires in round p in the execution Π_{sync} iff v fires during phase p in Π_{async} .

¹³In a follow-up work, we actually show that this assumption is necessary for the existence of synchronizers even when $L = 2$.

For simplicity of explanation, we assume that the network $\mathcal{N}_{\text{sync}}$ is deterministic. However, our scheme can easily capture randomized networks as well (i.e., by fixing the random bits in the synchronized simulation and feeding it to the async. one). See the full version for more details.

The Challenge. Consider a network of a threshold gate z with two incoming inputs: an excitatory neuron x , and an inhibitory neuron y . The weights are set such that z computes $X \wedge \bar{Y}$ thus it fires in round τ if x fired in round $\tau - 1$ and y did not fire. Implementing an $X \wedge \bar{Y}$ gate in the asynchronous setting is quite tricky. In the case where both x and y fire in round τ , in the synchronous network, z should not fire in round $\tau + 1$. However, in the asynchronous setting, if $\ell(x, z) < \ell(y, z)$, then z will mistakenly fire in round $\tau + \ell(x, z)$. This illustrates the need of enforcing a *delay* in the asynchronous simulation: the neurons should attempt firing only after receiving *all* their inputs from the previous phase. We handle this by introducing a pulse-generator module, that announces when it is safe to attempt firing.

To illustrate another source of challenge, consider the asynchronous implementation of an AND-gate $X \wedge Y$. If both x and y fire in round τ , then z fires in round $\tau + 1$ in the synchronous setting. However, if the latencies of the edges $\ell(x, z)$ and $\ell(y, z)$ are distinct, z receives the spike from x and y in *different* rounds, thus preventing the firing of z . Recall, that z has no memory, and thus its firing decision is based only on the potential level in the previous round. To overcome this hurdle, in the transformed network, each neuron in the original synchronous network is augmented with 3 copy-neurons, some of which have self-loops. Since self-loops have latency 1, once a neuron with a self-loop fires, it fires in the next round as well. This will make sure that the firing states of x and y are kept on being presented to z for sufficiently many rounds, which guarantees the existence of a round where both spikes arrive.

While solving one problem, introducing self-loops into the system brings along other troubles. Clearly, we would not want the neurons to fire forever, and at some point, those neurons should get inhibition to allow the beginning of a new phase. This calls for a delicate *reset* mechanism that cleans up the old firing states at the end of each phase, only after their values have already being used. Our final solution consists of global synchronization modules (e.g., pulse-generator, reset modules) that are inter-connected to a modified version of the synchronous network. Before explaining those constructions, we start by providing a modified neural timer $\text{DetTimer}_{\text{async}}$ adapted to asynchronous setting. This timer will be the basic building block in our global synchronization infrastructures.

Asynchronous Analog of DetTimer. A basic building block in our construction is a variant of DetTimer to the asynchronous setting. Observe that the DetTimer implementation of Sec. 2 might fail miserably in the asynchronous setting, e.g., when the edges $(a_{i-1,2}, a_{i,2})$ have latency 2 for every $i \geq 2$, and the remaining edges have latency 1, the timer will stop counting after $\Theta(\log t)$ rounds, rather than after t rounds. In the full version we show:

► **Lemma 11.** *[Neural Timer in the Asynchronous Setting] For a given time parameter t , there exists a deterministic network $\text{DetTimer}_{\text{async}}$ with $O(L \cdot \log t)$ neurons, satisfying that in the asynchronous setting with maximum latency L , the output neuron fires at least $\Theta(t)$ rounds, and at most $\Theta(L \cdot t)$ rounds after each firing of the input neuron.*

Description of the Synchronizer. The construction has two parts: a global infrastructure, that can be used to synchronize many networks¹⁴, and an adaptation of the given network $\mathcal{N}_{\text{sync}}$ into a network $\mathcal{N}_{\text{async}}$. The global infrastructure consists of the following modules:

- A *pulse generator* PG implemented by $\text{DetTimer}_{\text{async}}$ with time parameter $\Theta(L^3)$.
- A *reset* module R_1 implemented by a directed chain of $\Theta(L)$ neurons¹⁵ with input from the output neuron of the PG module.
- A *delay* module D implemented by $\text{DetTimer}_{\text{async}}$ with time parameter $\Theta(L^2)$ and input from the output of of the PG module.
- Another *reset* module R_2 implemented by a chain of $\Theta(L)$ neurons with input from D .

The heart of the construction is the pulse-generator that fires once within a fixed number of $\ell \in [\Theta(L^3), \Theta(L^4)]$ rounds, and invokes a cascade of activities at the end of each phase. When its output neuron g fires, it activates the reset and the delay modules, R_1 and D . The second reset module R_2 will be activated by the delay module D . Both reset modules R_1 and R_2 are implemented by chains of length L , with the last neuron on these chains being an *inhibitor* neuron. The role of the reset modules is to *erase* the firing states of some neurons (in $\mathcal{N}_{\text{async}}$) from the previous phase, hence their output neuron is an inhibitor. The timing of this clean-up is very delicate, and therefore the reset modules are separated by a delay module that prevents a premature operation. The total number of neurons in these global modules is $O(L \cdot \log L)$. We next consider the specific modifications to the synchronous network $\mathcal{N}_{\text{sync}}$ (see Fig. 3).

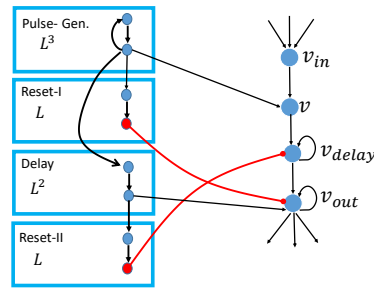
Modifications to the Network $\mathcal{N}_{\text{sync}}$. The input layer and output layer in $\mathcal{N}_{\text{async}}$ are *exactly* as in $\mathcal{N}_{\text{sync}}$. We will now focus on the set of *auxiliary* neurons V in $\mathcal{N}_{\text{sync}}$. In the network $\mathcal{N}_{\text{async}}$, each $v \in V$ is augmented by three additional neurons $v_{\text{in}}, v_{\text{delay}}$ and v_{out} . The incoming (resp., outgoing) neighbors to v_{in} (resp., v_{out}) are the out-copies (resp., in-copies) of all incoming (resp., outgoing) neighboring neurons of v . The neurons $v_{\text{in}}, v, v_{\text{delay}}$ and v_{out} are connected by a directed chain (in this order). Both v_{delay} and v_{out} have self-loops.

In case where the original network $\mathcal{N}_{\text{sync}}$ contains spiking neurons, the neuron v_{in} will be given the exact same firing function as v in Π_{sync} . That is, in phase p , v_{in} will be given the random coins¹⁶ used by v in round p in Π_{sync} . The other neurons v, v_{delay} and v_{out} are deterministic threshold gates. The role of the *out-copy* v_{out} is to *keep on presenting* the firing status of v from the previous phase $p - 1$ throughout the rounds of phase p . This is achieved through their self-loops. The role of the *in-copy* v_{in} is to simulate the firing behavior of v in phase p . We will make sure that v_{in} fires in phase p only if v fires in round p in Π_{sync} . For this reason, we set the incoming edge weights of v_{in} as well as its bias to be exactly the same as that of v in $\mathcal{N}_{\text{sync}}$. The neuron v is an AND gate of its in-copy v_{in} and the PG output g . Thus, we will make sure that v fires at the *end* of phase p only if v_{in} fires in this phase as well. The role of the delay copy v_{delay} is to delay the update of v_{out} to the up-to-date firing state of v (in phase p). Since both neurons v_{delay} and v_{out} have self-loops, at the end of each phase, we need to carefully reset their values (through inhibition). This is the role of the reset modules R_1 and R_2 . Specifically, the reset module R_1 operated by the pulse-generator inhibits v_{out} . The second reset module R_2 inhibits the delay neuron v_{delay} only after we can be certain that its value has already being “copied” to v_{out} . Finally, we describe the connections of the neuron v_{out} . The neuron v_{out} has an incoming edge from the reset module R_1 with a

¹⁴ It is indeed believed that the neural brain has centers of synchronization.

¹⁵ Each neuron in the chain has an incoming edge from its preceding neuron with weight 1 and threshold 1.

¹⁶ I.e., the random coins that are used to simulate the firing decision of v .



■ **Figure 3** Illustration of the synchronizer modules. Left: global modules implemented by neural timers. Right: a neuron $v \in N_{\text{sync}}$ augmented by three additional neurons that interact with the global modules.

super-large weight. This makes sure that when the reset module is activated, v_{out} will be inhibited shortly after. In addition, it has a self-loop also of large weight (yet smaller than the inhibition edge) that makes sure that if v_{out} fires in a given round, and the reset module R_1 is not active, v_{out} also fires in the next round. Lastly, if v_{out} did not fire in the previous round, then it fires when receiving the spikes from *both* the delay module and from the delay copy v_{delay} . This will make sure that the firing state of v_{delay} will be copied to v_{out} only after the output of the delay module D fires. The complete analysis is given in the full version.

5 Open Problems

In this paper we introduce the problems of neural timer and neural counter in order to shed light into the way that neurons measure time in real biological neural networks. We believe that these timer and counting modules should be useful for many other computational tasks. The key application considered in this paper is for asynchronous computation. For that purpose we introduce a simplified asynchronous model. It would be interesting to delve into this setting and tighten the overhead in the number of neurons and computation time. Finally, exploring the connections between succinct neural networks and dynamic streaming algorithms is yet another promising research direction. The approximate counting problem already provides a positive indication for a potential relation between these models.

References

- 1 Edgar D Adrian. The impulses produced by sensory nerve endings. *The Journal of physiology*, 61(1):49–72, 1926.
- 2 Melissa J Allman, Sundeep Teki, Timothy D Griffiths, and Warren H Meck. Properties of the internal clock: first-and second-order principles of subjective time. *Annual review of psychology*, 65:743–771, 2014.
- 3 Douglas B Armstrong, Arthur D Friedman, and Premachandran R Menon. Design of asynchronous circuits assuming unbounded gate delays. *IEEE Transactions on Computers*, 100(12):1110–1120, 1969.
- 4 Baruch Awerbuch and David Peleg. Network Synchronization with Polylogarithmic Overhead. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 514–522, 1990.
- 5 Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys (CSUR)*, 38(1):1, 2006.

- 6 Chi-Ning Chou, Kai-Min Chung, and Chi-Jen Lu. On the Algorithmic Power of Spiking Neural Networks. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, pages 26:1–26:20, 2019.
- 7 RE Lee DeVille and Charles S Peskin. Synchrony and asynchrony in a fully stochastic neural network. *Bulletin of mathematical biology*, 70(6):1608–1633, 2008.
- 8 Huawei Fan, Yafeng Wang, Hengtong Wang, Ying-Cheng Lai, and Xingang Wang. Autapses promote synchronization in neuronal networks. *Scientific reports*, 8(1):580, 2018.
- 9 Gerald T Finnerty, Michael N Shadlen, Mehrdad Jazayeri, Anna C Nobre, and Dean V Buonomano. Time in cortical circuits. *Journal of Neuroscience*, 35(41):13912–13916, 2015.
- 10 Philippe Flajolet. Approximate Counting: A Detailed Analysis. *BIT*, 25(1):113–134, 1985.
- 11 Wulfram Gerstner, Andreas K Kreiter, Henry Markram, and Andreas VM Herz. Neural codes: firing rates and beyond. *Proceedings of the National Academy of Sciences*, 94(24):12740–12741, 1997.
- 12 Scott Hauck. Asynchronous design methodologies: An overview. *Proceedings of the IEEE*, 83(1):69–93, 1995.
- 13 Kaori Ikeda and John M Bekkers. Autapses. *Current Biology*, 16(9):R308, 2006.
- 14 Fabian Kuhn, Joel Spencer, Konstantinos Panagiotou, and Angelika Steger. Synchrony and asynchrony in neural networks. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete algorithms*, pages 949–964. SIAM, 2010.
- 15 Robert A. Legenstein, Wolfgang Maass, Christos H. Papadimitriou, and Santosh Srinivas Vempala. Long Term Memory and the Densest K-Subgraph Problem. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 57:1–57:15, 2018.
- 16 Benjamin Lindner. Some unsolved problems relating to noise in biological systems. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(01):P01008, 2009.
- 17 Nancy Lynch and Cameron Musco. A Basic Compositional Model for Spiking Neural Networks. *arXiv preprint*, 2018. [arXiv:1808.03884](https://arxiv.org/abs/1808.03884).
- 18 Nancy Lynch, Cameron Musco, and Merav Parter. Computational Tradeoffs in Biological Neural Networks: Self-Stabilizing Winner-Take-All Networks. In *Proceedings of the 8th Conference on Innovations in Theoretical Computer Science (ITCS)*, 2017.
- 19 Nancy Lynch, Cameron Musco, and Merav Parter. Spiking Neural Networks: An Algorithmic Perspective. In *5th Workshop on Biological Distributed Algorithms (BDA 2017)*, July 2017.
- 20 Nancy A. Lynch, Cameron Musco, and Merav Parter. Neuro-RAM Unit with Applications to Similarity Testing and Compression in Spiking Neural Networks. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 33:1–33:16, 2017.
- 21 Jun Ma, Xinlin Song, Wuyin Jin, and Chuni Wang. Autapse-induced synchronization in a coupled neuronal network. *Chaos, Solitons & Fractals*, 80:31–38, 2015.
- 22 Wolfgang Maass. Lower Bounds for the Computational Power of Networks of Spiking Neurons. *Electronic Colloquium on Computational Complexity (ECCC)*, 1(19), 1994. URL: <http://eccc.hpi-web.de/eccc-reports/1994/TR94-019/index.html>.
- 23 Wolfgang Maass. On the computational power of noisy spiking neurons. In *Advances in Neural Information Processing Systems 8 (NIPS)*, 1996.
- 24 Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.
- 25 Rajit Manohar and Yoram Moses. The eventual C-element theorem for delay-insensitive asynchronous circuits. In *2017 23rd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 102–109. IEEE, 2017.
- 26 Hugo Merchant, Deborah L Harrington, and Warren H Meck. Neural basis of the perception and estimation of time. *Annual review of neuroscience*, 36:313–336, 2013.
- 27 Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.

- 28 Christos H Papadimitriou and Santosh S Vempala. Random projection in the brain and computation with assemblies of neurons. In *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- 29 Jens Sparsø. Asynchronous circuit design-a tutorial. In *Chapters 1-8 in “Principles of asynchronous circuit design-A systems Perspective”*. Kluwer Academic Publishers, 2001.
- 30 Misha V Tsodyks and Henry Markram. The neural code between neocortical pyramidal neurons depends on neurotransmitter release probability. *Proceedings of the national academy of sciences*, 94(2):719–723, 1997.
- 31 Barbeeba Wang and Nancy Lynch. Integrating Temporal Information to Spatial Information in a Neural Circuit. *arXiv preprint*, 2019. [arXiv:1903.01217](https://arxiv.org/abs/1903.01217).

Triconnected Planar Graphs of Maximum Degree Five are Subhamiltonian*

Michael Hoffmann 

Department of Computer Science, ETH Zürich, Zürich, Switzerland
hoffmann@inf.ethz.ch

Boris Klemz 

Institute of Computer Science, Freie Universität Berlin, Berlin, Germany
klemz@inf.fu-berlin.de

Abstract

We show that every triconnected planar graph of maximum degree five is subhamiltonian planar. A graph is subhamiltonian planar if it is a subgraph of a Hamiltonian planar graph or, equivalently, if it admits a 2-page book embedding. In fact, our result is stronger because we only require vertices of a separating triangle to have degree at most five, all other vertices may have arbitrary degree. This degree bound is tight: We describe a family of triconnected planar graphs that are not subhamiltonian planar and where every vertex of a separating triangle has degree at most six. Our results improve earlier work by Heath and by Bauernöppel and, independently, Bekos, Gronemann, and Raftopoulou, who showed that planar graphs of maximum degree three and four, respectively, are subhamiltonian planar. The proof is constructive and yields a quadratic time algorithm to obtain a subhamiltonian plane cycle for a given graph.

As one of our main tools, which might be of independent interest, we devise an algorithm that, in a given 3-connected plane graph satisfying the above degree bounds, collapses each maximal separating triangle into a single edge such that the resulting graph is biconnected, contains no separating triangle, and no separation pair whose vertices are adjacent.

2012 ACM Subject Classification Mathematics of computing → Paths and connectivity problems; Mathematics of computing → Graphs and surfaces; Theory of computation → Computational geometry; Human-centered computing → Graph drawings

Keywords and phrases Graph drawing, book embedding, Hamiltonian graph, planar graph, bounded degree graph, graph augmentation, computational geometry, SPQR decomposition

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.58

Funding *Michael Hoffmann*: Research supported by the Swiss National Science Foundation within the collaborative DACH project *Arrangements and Drawings* as SNSF Project 200021E-171681.

Acknowledgements This research was initiated during the Geometric Graphs Week 2016 at UPC, Barcelona, and reinvigorated at the Lorentz Center Workshop on Fixed-Parameter Computational Geometry 2018 in Leiden. We thank the organizers and participants for the stimulating atmosphere and, in particular, Oswin Aichholzer, Bahareh Banyassady, Philipp Kindermann, Irina Kostitsyna, and Fabian Lipp for initial discussions about the problem.

1 Introduction

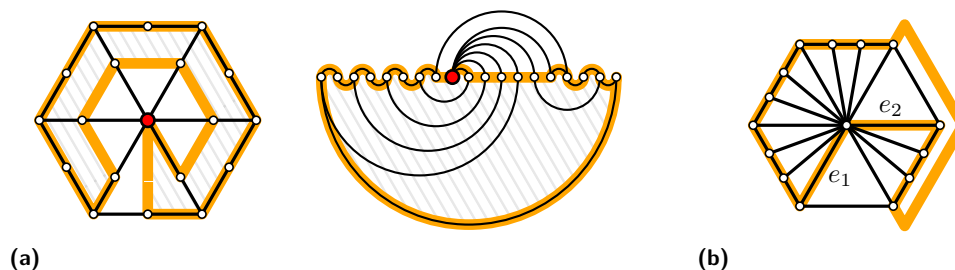
The structure of spanning a.k.a. Hamiltonian paths and cycles in graphs has been a fruitful subject of intense research over many decades, both from a combinatorial and from an algorithmic point of view. For general graphs, sufficient conditions for the existence of a Hamiltonian cycle typically involve rather strong assumptions on the degree, such as in

* Due to space constraints, some proofs in this extended abstract are only sketched or omitted entirely.



Dirac's Theorem [9] (minimum degree $\geq n/2$), Ore's Theorem [16] (degree sum of every nonadjacent vertex pair $\geq n$), or Asratian and Khachatrian's Theorem [1] ($\deg(u) + \deg(w) = |N(u) + N(v) + N(w)|$, for every induced path uvw). Planar graphs provide a lot more structure so that by a famous theorem of Tutte, 4-connectivity suffices to guarantee the existence of a Hamiltonian cycle [19], which can be computed in linear time [7]. In contrast, deciding Hamiltonicity is NP-complete for 3-connected cubic planar graphs [11]. Finally, *maximal* planar graphs of degree at most six are Hamiltonian [10]. We observe that both vertex degree and connectivity are crucial parameters concerning Hamiltonicity.

Hamiltonian cycles are also of interest in the context of graph embeddings. Specifically, in a *book embedding*, all vertices are embedded on a line called *spine*, and every edge is embedded in a halfplane, called *page*, bounded by the spine. No two edges (on the same page) cross. If k pages are used, then the corresponding embedding is a k -page book embedding. Note that a k -page book embedding with $k \leq 2$ is plane. Bernhart and Kainen [4] characterized those graphs that can be embedded on k pages, for $k \leq 2$. For $k = 2$ these are the subhamiltonian planar¹ graphs, that is, subgraphs of Hamiltonian planar graphs, cf. Figure 1a. Hence not all planar graphs can be embedded on two pages; in fact, it is NP-complete to decide whether a given planar graph is subhamiltonian [20], even if all vertices have degree at most seven [2]. However, no planar graph is too far away from being subhamiltonian: Subdividing at most $n/2$ of the up to $3n - 6$ edges of a planar graph on n vertices yields a subhamiltonian planar graph [6].



■ **Figure 1** (a) A nonhamiltonian graph with a subhamiltonian cycle and a corresponding two-page book embedding. (b) A subhamiltonian cycle using e_1 and e_2 in a wheel with at least four spokes.

Despite a plethora of results concerned with Hamiltonian cycles in planar graphs and book embeddings, several fundamental questions are still open. Let us give just two prominent examples to illustrate this point. For once, there is Barnette's Conjecture: "Every 3-connected cubic bipartite planar graph is Hamiltonian." And then there is the question if every planar graph can be embedded on three pages. Yannakakis showed, improving a series of earlier results, that four pages are sufficient for every planar graph [22]. However, a corresponding lower bound is still elusive, in spite of initial claims [21].

In this paper, we investigate what upper bound on the vertex degree guarantees that a planar graph is subhamiltonian planar. Heath [14] showed that maximum degree three suffices. Later, Bauernöppel [2] and, independently, Bekos et al. [3] showed that every planar graph of degree at most four is subhamiltonian planar. On the negative side, Bauernöppel [2] described planar graphs of vertex degree at most seven that are not subhamiltonian (these graphs are biconnected but not 3-connected). In a preprint, Guan and Yang [12] show that planar graphs of maximum degree five can be embedded on three pages. Our goal is to further relax the degree bound so as to (ultimately) determine what is the largest k so that every

¹ We prefer the term "subhamiltonian planar" over just "subhamiltonian" because the former is more descriptive as a shortcut for "subgraph of a Hamiltonian planar graph".

planar graph of degree at most k is subhamiltonian planar. Specifically, we are interested in the case where the given graph is 3-connected and hence the combinatorial embedding is unique. Along these lines, we make a natural next step by considering the case of maximum degree five. In fact, we prove a much stronger statement where the degree restriction applies to vertices of separating 3-cycles only.

► **Theorem 1.** *Let G be a 3-connected simple planar graph on n vertices where every vertex of a separating 3-cycle has degree at most five. Then G is subhamiltonian planar. Moreover, a subhamiltonian plane cycle for G can be computed in $O(n^2)$ time.*

► **Corollary 2.** *Every 3-connected simple planar graph with maximum vertex degree five can be embedded on two pages, and such an embedding can be computed in quadratic time.*

We also show that the degree bound in Theorem 1 is tight.

► **Theorem 3.** *There exists a family of 3-connected simple planar graphs that are not subhamiltonian planar and where every vertex of a separating 3-cycle has degree at most six.*

Organization. We begin by introducing some terminology in Section 2. In Section 3, we study three special cases for which Theorem 1 is easily proven. In Section 4, we proceed with a high-level overview of the proof of Theorem 1, which is then developed in some more detail in Sections 5–7. We conclude with an example to illustrate Theorem 3 in Section 8.

2 Notation

All graphs in this paper are undirected. We denote by $V(G)$ the vertex set and by $E(G)$ the edge set of a graph G . For a set of edges $E \subseteq E(G)$ we use $V(E)$ to denote the set of vertices that are incident to at least one edge in E . For a face f of G we denote by ∂f the closed walk in G that traverses the vertices and edges on the boundary of f in anticlockwise direction. If G is biconnected, then ∂f is a cycle, for every face f of G . A *Hamiltonian* cycle for a graph is a simple cycle through all vertices and a graph is *Hamiltonian* if it contains a Hamiltonian cycle. An *augmentation* of a graph $G = (V, E)$ is a supergraph $A = (V, E')$ with $E' \supseteq E$. If G is a plane graph, then a *plane augmentation* H of G is an augmentation that respects the embedding of G , that is, if Γ denotes the embedding of G and Γ' denotes the embedding of H , then $\Gamma'|_G = \Gamma$. A *subhamiltonian cycle* for a graph G is a Hamiltonian cycle in some augmentation of G .

We distinguish between separating 3-cycles as a notion for both abstract and embedded graphs and separating triangles in embedded graphs. A *separating 3-cycle* is a 3-cycle whose removal disconnects the graph. A *separating triangle* is a 3-cycle C of an embedded graph G such that both the interior and the exterior region bounded by C contain some vertex of G .

For a cycle C in a plane graph G denote by G_C^+ the plane subgraph of G that contains all vertices and edges on C and exterior to C . Similarly denote by G_C^- the plane subgraph of G that contains all vertices and edges on C and interior to C . The *inside* of C refers to the interior of the bounded region enclosed by C . So a vertex of G *inside of* C is a vertex of $G_C^- \setminus C$. Analogously a vertex *outside of* C is a vertex of $G_C^+ \setminus C$. A separating triangle T is called *trivial* if $G_T^- \simeq K_4$ and *nontrivial*, otherwise.

Our algorithm uses a decomposition of the graph into its triconnected components, which can be efficiently maintained via the SPQR-tree data structure [13, 15]. We use the terminology by Gutwenger and Mutzel [13].

3 Three simple cases

It suffices to prove Theorem 1 for an arbitrary plane embedding of the given graph, which we assume to be represented as a doubly-connected edge list (DCEL) [8]. In fact, by 3-connectivity, the combinatorial plane embedding is unique up to the choice of the outer face, and there is no difference between separating 3-cycles and separating triangles. So let G be an embedded 3-connected simple planar graph (with a fixed outer face) where every vertex incident to a separating triangle has degree at most five. By combining known results, it is easy to deal with the case that G has no separating triangle:

► **Theorem 4.** *If an embedded simple planar graph $\mathcal{G} = (V, E)$ does not contain any separating triangle, then for any two distinct edges $e_1, e_2 \in E$ there is a plane augmentation \mathcal{H} of \mathcal{G} that contains a Hamiltonian cycle C using e_1 and e_2 . Moreover, the cycle C can be computed in $O(n^2)$ time, where n is the number of vertices in \mathcal{G} . If both e_1 and e_2 are on the outer face of \mathcal{G} , then C can be computed in linear time.*

Proof Sketch. Augment \mathcal{G} using an algorithm by Biedl, Kant, and Kaufmann [5] to obtain, if possible, a 4-connected planar graph, which contains the desired Hamiltonian cycle due to a theorem of Sanders [17, Corollary 2]. If such an augmentation is impossible, then the graph can be augmented to a wheel, in which the desired cycle is easily found; see Figure 1b.

Algorithmically, the bottleneck is Sanders' theorem. While the original formulation is purely existential, a recently developed algorithmic version by Schmid and Schmidt [18] allows to compute such a cycle in quadratic time. For the special case where both e_1 and e_2 are on the outer face, we can use the linear time algorithm by Chiba and Nishizeki [7]. ◀

In order to be able to argue inductively, we prove a stronger statement than necessary, namely a version of Theorem 1 where, similar to the statement in Theorem 4, two edges of the desired subhamiltonian cycle may be prescribed:

► **Theorem 5.** *Let $\mathcal{G} = (V, E)$ be an embedded 3-connected simple planar graph on n vertices with outer face T_o , where every vertex that is incident to a separating triangle has degree at most five. Further, let $F \subset E$ be a set of at most two edges so that*

(P1) *all edges in F are edges of T_o ;*

(P2) *if T_o is not a triangle, then $F = \emptyset$;*

(P3) *if $F \neq \emptyset$, then no vertex of T_o is incident to a separating triangle of \mathcal{G} ; and*

(P4) *if $F \neq \emptyset$, then either at least one vertex of T_o has degree three in \mathcal{G} , or all vertices of T_o have degree four in \mathcal{G} .*

Then there is a plane augmentation of \mathcal{G} that contains a Hamiltonian cycle C that uses all edges from F . Moreover, the cycle C can be computed in $O(n^2)$ time.

The proof, sketched in Sections 5–7, is carried out by induction on the number of vertices. In this context, one can easily show that:

► **Lemma 6.** *Suppose that the statement of Theorem 5 holds for all graphs with at most $n - 1$ vertices, where $n \geq 6$. Then it also holds for every graph on n vertices that contains at least one nontrivial separating triangle.*

Proof Sketch. Let T be a nontrivial separating triangle in \mathcal{G} . Replace the interior of T with a single vertex d and inductively find a subhamiltonian cycle C in the resulting smaller graph. Then, inductively obtain a subhamiltonian cycle C' for \mathcal{G}_T^- . For this second inductive call, we make use of the ability to prescribe edges, depending on how C visits d , in order to ensure that the cycles C and C' may be glued together to obtain a subhamiltonian cycle for \mathcal{G} . ◀

The degree restriction basically enforces that the separating triangles of G are pairwise vertex-disjoint. However, there are some exceptional configurations where two separating triangles share an edge. Our next goal is to classify these configurations precisely.

A *double kite* is a subgraph $U \simeq K_4$ of an embedded graph \mathcal{G} so that exactly two of the four triangles of U are separating in \mathcal{G} . The two separating triangles are said to *define* the double kite. Note that \mathcal{G} may contain multiple double kites, see Figure 2a. We refer \mathcal{G} itself as a *trivial double kite* if it is 3-connected, contains a double kite, and has precisely 6 vertices.

► **Lemma 7.** *Let \mathcal{G} be an embedded 3-connected simple planar graph and let T_1, T_2 be two distinct separating triangles of \mathcal{G} such that every vertex incident to T_1 has degree at most five. Then if T_1 and T_2 share a vertex, the triangles T_1 and T_2 define a double kite.*

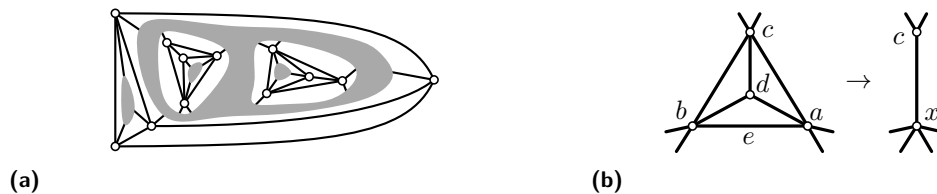
► **Observation 8.** *Let \mathcal{G} be an embedded 3-connected simple planar graph and let T_1, T_2 be two distinct trivial separating triangles of \mathcal{G} such that all vertices incident to T_1 or T_2 have degree at most five. Then if T_1 and T_2 share a vertex, the graph \mathcal{G} is a trivial double kite.*

For a trivial double kite G , the statement of Theorem 5 is easily verified. Hence, from now on we may assume that the separating triangles of G are trivial and pairwise vertex-disjoint.

4 Proof overview

To prove Theorem 5 we proceed in three steps.

In the first step (Section 5), we destroy all separating triangles using edge collapses. An edge $e = ab$ of a separating triangle $T = abc$ is *collapsed* by contracting (the three edges of) the triangle that is spanned by the endpoints of e and the single vertex d inside T into a new vertex x and merging the two edges ac and bc to a new edge cx ; see Figure 2b. Observe that an edge collapse can be performed in constant time in a DCEL.



■ **Figure 2** (a) A graph with three double kites. (b) Collapsing the edge $e = ab$.

A collapse operation may create multiple edges if a and b have a common neighbor $z \notin \{c, d\}$. However, we will assert that an edge ab is collapsed only if the triangle abz is not separating, i.e., one of its sides is empty. In particular, if T is not part of a double kite, we may collapse any of its edges by Lemma 7. Our assertion ensures that whenever an edge collapse creates multiple edges, we may merge them into a singleton edge without losing information about the embedding. Hence, we may assume that the graph resulting from an edge collapse is always simple. Moreover, note that the collapse operation does not increase the degree of any vertex. In particular, the degree of c decreases by two. The degree of the new vertex x is at most five. Hence, collapsing an edge in G results in a graph that satisfies the degree constraints, unless the collapse creates a new separating triangle.

We describe a procedure to find a set $K \subset E$ of edges such that the graph G' obtained by simultaneously collapsing all edges of K does not contain any separating triangle.

In the second step (Section 6), we augment G' by *stellating* every face, that is, for each nontriangular face f of G' we insert a new vertex v_f into f and we add an edge between v_f and each vertex on the boundary of f . By choosing the set K suitably, we ensure that the graph G'' resulting from these stellations does not contain any separating triangle. Thus, using Theorem 4 we obtain a subhamiltonian cycle C'' for G'' .

Finally, in the third step (Section 7), we iteratively revert the edge collapses. Due to the stellated faces, we have some control over the possible ways that C'' visits the vertex pairs in G'' that result from collapsing a triangle of G . We will show that a series of local rerouting steps suffices to transform C'' into a subhamiltonian cycle for G .

5 Collapsing edges

Recall that G is an embedded 3-connected simple planar graph in which every separating triangle is trivial and all vertices incident to a separating triangle have degree at most five. Moreover, G is not a trivial double kite. Let S denote the set of separating triangles of G , which are pairwise vertex-disjoint by Observation 8. We want to find a set $K \subset E$ of edges so that (1) every edge in K is incident to a triangle in S ; (2) every triangle in S is incident to exactly one edge from K ; and (3) the graph obtained by simultaneously collapsing the edges of K does not contain any separating triangle.

Obviously, there exists a set \hat{K} of edges that satisfies (1) and (2). Suppose that (3) does not hold for \hat{K} , that is, the graph \hat{G} obtained from G by collapsing the edges in \hat{K} contains a separating triangle T . By (2) we know that T is not a triangle in G , but T corresponds to a separating k -cycle in G , for $k \geq 4$. By (1), (2), and since the triangles in S are pairwise vertex-disjoint, at most every other edge of a cycle in G is in \hat{K} and, therefore, we have $k \leq 6$. In other words, every separating triangle in \hat{G} corresponds to a separating k -cycle in G where $k \in \{4, 5, 6\}$ and exactly $k - 3$ edges are in \hat{K} . Moreover, for any such separating k -cycle in G , both the interior and the exterior must contain at least one vertex that is not the interior vertex of a triangle from S because by (2) every interior vertex of a triangle from S disappears in some collapse.

Inspired by these observations, we call a cycle C of an embedded graph *hyperseparating* if both the interior and the exterior contain at least one vertex that is not the interior vertex of a trivial separating triangle. We define an *inhibitor* to be a hyperseparating k -cycle I , where $k \in \{4, 5, 6\}$, for which at least $k - 3$ edges are incident to a trivial separating triangle, for an illustration refer to Figure 3d. We refer to these at least $k - 3$ edges as *constrained*. An edge e of an embedded graph is *constrained* if there exists an inhibitor I so that e is a constrained edge of I and *unconstrained*, otherwise. An inhibitor of length $k \in \{4, 5, 6\}$ is also called a *k-inhibitor*.

Note that a set \hat{K} of edges that satisfies (1) and (2) also satisfies (3) if for every inhibitor of length k in G , no more than $k - 4$ of its constrained edges are in \hat{K} . Next we study the:

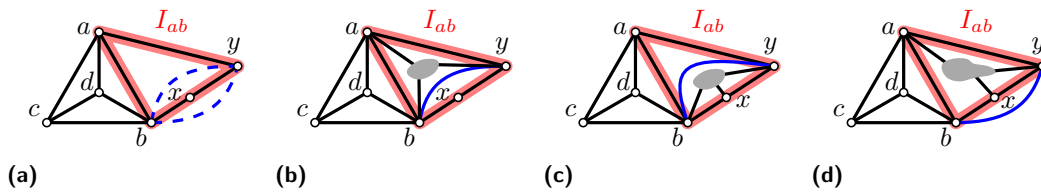
5.1 Structure of inhibitors

First, let us observe that 4-inhibitors of G are chordless.

► **Observation 9.** *Let \mathcal{G} be an embedded 3-connected simple planar graph and let $T = abc$ be a trivial separating triangle of \mathcal{G} with inner vertex d such that all vertices incident to T have degree at most five. Further, let $abxy$ be a 4-inhibitor constraining ab . Then $\{x, y\} \cap \{c, d\} = \emptyset$.*

► **Lemma 10.** *Let \mathcal{G} be an embedded 3-connected simple planar graph and let $T = abc$ be a trivial separating triangle of \mathcal{G} with inner vertex d such that all vertices incident to T have degree at most five. Suppose that ab is constrained by a 4-inhibitor $I_{ab} = abxy$. Then I_{ab} is chordless, unless ab is incident to two separating triangles that define a double kite.*

Proof. Assume without loss of generality that by is a chord of I_{ab} . By Observation 9, $x, y \notin \{c, d\}$ and, so, the degree of b is saturated; see Figure 3a. We claim that by is on the side of I_{ab} that contains c and d . To see this, assume the contrary. Since I_{ab} is separating, there is some vertex z on the side of I_{ab} that does not contain c, d . The chord by splits this side of I_{ab} into two triangles, one of which contains z , see Figure 3b and 3c. However, by 3-connectivity, this implies that b has a neighbor in this triangle, in contradiction to the degree bound of b . So the claim holds, that is, by is indeed on the side of I_{ab} that contains c, d . Then aby is a triangle that separates x from c, d , see Figure 3d. By Lemma 7, the triangles abc and aby define a double kite. ◀

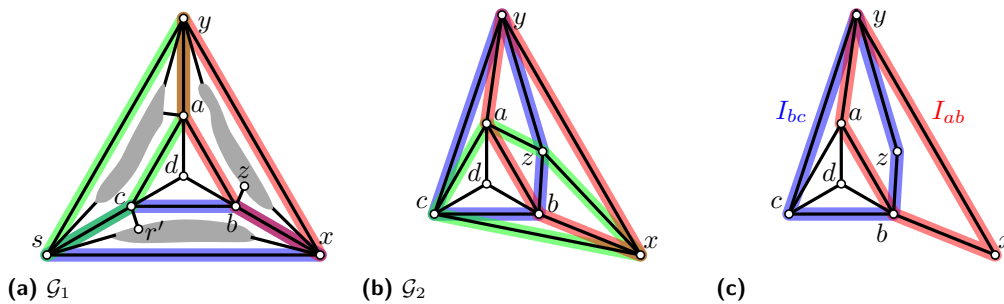


■ **Figure 3** A 4-inhibitor $I_{ab} = abxy$ that constrains the edge ab of the separating triangle abc .

A necessary requirement for the existence of a set K of edges to collapse is that our graph does not contain a separating triangle for which every edge is constrained by a 4-inhibitor. We show that this requirement is met, except for some specific special cases.

► **Lemma 11.** *Let \mathcal{G} be an embedded 3-connected simple planar graph and let \mathcal{I} be a set of 4-inhibitors of \mathcal{G} . Further, let T be a trivial separating triangle of \mathcal{G} such that all vertices incident to T have degree at most five. Finally, assume that \mathcal{G} contains no separating triangle that together with T defines a double kite.*

Then either (1) T is incident to at least one edge that is not constrained by a 4-inhibitor of \mathcal{I} ; or (2) \mathcal{G} contains a subgraph \mathcal{G}' that is isomorphic to one of the graphs $\mathcal{G}_1, \mathcal{G}_2$ depicted in Figure 4 such that each thick (colored) edge of \mathcal{G}' is incident to some 4-inhibitor of \mathcal{I} .



■ **Figure 4** If all three edges of a separating triangle in the graph \mathcal{G} are constrained by a 4-inhibitor, there is a subgraph \mathcal{G}' isomorphic to \mathcal{G}_1 or \mathcal{G}_2 . The gray parts in (a) represent arbitrary subgraphs.

Proof Sketch. Assume that each edge of $T = abc$ is constrained by a 4-inhibitor of \mathcal{I} , and that \mathcal{G} contains no subgraph \mathcal{G}' as in the claim. We denote the inhibitors of \mathcal{I} constraining ab , bc , and ac by $I_{ab} = abxy$, $I_{bc} = bcst$, and $I_{ac} = acvw$, respectively. By Lemma 10, we have $c, d \notin I_{ab}$, $a, d \notin I_{bc}$, and $b, d \notin I_{ac}$, where d denotes the inner vertex of T . By 3-connectivity and since I_{ab} is separating, we may assume w.l.o.g. that b has a neighbor z which is located on the side of I_{ab} that does not contain c ; see Figure 4c. Due to the degree bound of b , we have $t \in \{x, z\}$.

First, assume that $t = z$. By planarity, s belongs to I_{ab} ; and by Lemma 10 applied to I_{bc} we have $s = y$ and, thus, $I_{bc} = bcyz$. We study the third inhibitor $I_{ac} = acvw$. We have $v \neq y$, as otherwise ay would be a chord of I_{ac} , contradicting Lemma 10. The triangle acy is nonseparating by Lemma 7 and the assumption that T does not define a double kite. Thus, v is located on the side of I_{bc} that does not contain a and, hence, by planarity, v or w has to belong to I_{bc} . Since $c \in I_{ac} \cap I_{bc}$, Lemma 10 implies that the only other vertex of I_{ac} in this intersection is z . As z and c are located on distinct sides of I_{ab} , the vertex v must belong to I_{ab} and, thus, $v = x$ and $I_{ac} = acxz$. Altogether, this establishes that \mathcal{G} contains a subgraph $\mathcal{G}' \simeq \mathcal{G}_2$ as in the statement, which yields the desired contradiction.

It remains to consider the case that $t = x$. Using similar arguments as in the previous paragraph, we obtain a contradiction to the assumption that \mathcal{G} does not contain a subgraph $\mathcal{G}'' \simeq \mathcal{G}_1$ as in the statement. \blacktriangleleft

Similar to the proof of Lemma 6, we can deal with the special case that there is a separating triangle in S such that each of its edges is constrained using the inductive framework of the proof of Theorem 5. So assume from now on that G does not contain a subgraph isomorphic to \mathcal{G}_1 or \mathcal{G}_2 as in Lemma 11. Our plan is to use Lemma 11 to identify an unconstrained edge, which is then collapsed. This procedure is then iterated until no separating triangle is left. The edges collapsed during this process form the desired set K .

Due to the degree bound, it is easy to determine in quadratic time the set of *ineligible* edges, that is, edges of separating triangles that are constrained by 4-inhibitors. Whenever an edge collapses, we check all separating triangles in the constant size neighborhood in order to update the set of ineligible edges in constant time. These edges are never considered for inclusion into the set \mathcal{K} of edges to collapse.

An edge collapse is *3-connectivity preserving* if the graph resulting from the collapse is 3-connected. As long as there is an eligible edge whose collapse preserves 3-connectivity, we collapse it. In order to test whether this is the case for an edge ab of a separating triangle $T = abc$, it suffices to determine and check all pairs of distinct faces f_a and f_b of \mathcal{G}_T^+ incident to a and b , respectively, and a vertex $v \neq c$ such that $v \in \partial f_a \cap \partial f_b$. This test can be performed in linear time, given that there is only a constant number of choices for f_a and f_b . Observe that negative test results are robust under 3-connectivity preserving collapses; whereas, in general, positive results are not. Hence, it suffices to consider every eligible edge at most once, and so the time spent on these tests is quadratic overall.

However, in general an edge collapse may reduce the connectivity of the graph. In this case, we plan to recurse on the triconnected components of the graph. To make such a recursion work in the context of our overall proof strategy, we must take special care concerning the vertices of separation pairs. Specifically, as we will discuss in the following section, we should never create a separation pair whose vertices are adjacent.

5.2 Avoiding adjacent separation pairs

Recall that we plan to stellate each face of the graph G' that is obtained by simultaneously collapsing all edges in K , and that we need to ensure that the resulting graph G'' does not contain any separating triangle. Consider a face f of G' and assume that its stellation creates a separating triangle $s = av_f$ where v_f is the new vertex inserted into f . Note that the vertices a and b are incident to f . Therefore, the edge $ab \in E(G')$ is a chord of ∂f and, moreover, a, b is a separation pair of G' .

In order to avoid this situation, it suffices to choose the set K subject to the following additional constraint: (4) the graph obtained by simultaneously collapsing the edges of K does not create an *adjacent* separation pair, i.e., a separation pair a, b where a and b are adjacent. Inspired by this observation, we devise a strengthened version (Lemma 13) of Lemma 11. For its proof, we require the following observation:

► **Observation 12.** *Let \mathcal{G} be an embedded 3-connected simple planar graph and let $T = abc$ be a trivial separating triangle of \mathcal{G} with inner vertex d . Further, assume that collapsing ab results in a graph \mathcal{G}' that is 2-connected but not 3-connected; and let p, q be a separation pair of \mathcal{G}' . Then we may assume that p is the new vertex that is created by the collapse of ab and that $q \notin \{c, d\}$.*

► **Lemma 13.** *Let \mathcal{G} be an embedded 3-connected simple planar graph and let \mathcal{I} be a set of 4-inhibitors of \mathcal{G} . Further, let $T = abc$ be a trivial separating triangle of \mathcal{G} with inner vertex d such that all vertices incident to T have degree at most five. Finally, assume that \mathcal{G} contains no separating triangle that together with T defines a double kite.*

Then either (1) T is incident to at least one edge e such that (i) e is not constrained by a 4-inhibitor of \mathcal{I} ; and (ii) collapsing e does not create an adjacent separation pair; or (2) \mathcal{G} contains a subgraph \mathcal{G}' that isomorphic to one of the graphs $\mathcal{G}_1, \mathcal{G}_2$ depicted in Figure 4 such that each thick (colored) edge of \mathcal{G}' is incident to some 4-inhibitor of \mathcal{I} .

5.3 Choosing the set of edges to collapse

We are now prepared to discuss how the desired edge set K is obtained.

► **Theorem 14.** *Let \mathcal{H} be an embedded 3-connected simple planar graph on n vertices where every vertex incident to a separating triangle has degree at most five and where every separating triangle is trivial. Further, let \mathcal{S} denote the set of separating triangles in \mathcal{H} . Assume that \mathcal{H} is not a trivial double kite and that \mathcal{H} contains no subgraph isomorphic to \mathcal{G}_1 or \mathcal{G}_2 . Then we can compute in $O(n^2)$ time a set $\mathcal{K} \subset E(\mathcal{H})$ of edges so that:*

- (I1) every edge in \mathcal{K} is incident to a triangle in \mathcal{S} ;
- (I2) every triangle in \mathcal{S} is incident to exactly one edge from \mathcal{K} ;
- (I3) \mathcal{H} contains no k -inhibitor I such that \mathcal{K} contains $k - 3$ or more edges of I ; and
- (I4) the graph \mathcal{H}'' obtained by simultaneously collapsing the edges of \mathcal{K} is biconnected and does not contain an adjacent separation pair.

Proof Sketch. The proof is by induction on the number $|\mathcal{S}|$ of separating triangles. Let e be an edge of \mathcal{H} that satisfies Property (1) of Lemma 13. Let \mathcal{H}' be the embedded simple planar graph obtained by collapsing e in \mathcal{H} . We show that \mathcal{H}' is not a trivial double kite or contains a subgraph isomorphic to \mathcal{G}_2 . We also show that e can always be chosen such that \mathcal{H}' contains no \mathcal{G}_1 . If \mathcal{H}' is 3-connected, we inductively obtain an edge set \mathcal{K}' that satisfies the Properties (I1)–(I4) and, hence, $\mathcal{K} = \mathcal{K}' \cup \{e\}$ satisfies these properties for \mathcal{H} .

It remains to deal with the case that for every edge e that satisfies Property (1) of Lemma 13, collapsing e results in a graph \mathcal{H}' that is biconnected, but not 3-connected. In this case, we recurse on the triconnected components of \mathcal{H}' . In particular, from the fact that \mathcal{H}' contains no adjacent separation pair, we may derive that each separating triangle of \mathcal{H}' appears, with all its real edges, in some rigid triconnected component R of \mathcal{H}' , which is a simple 3-connected planar graph. The graph R may contain separating triangles that do not belong to \mathcal{S} , namely the separating triangles that are incident to virtual edges. Keeping in mind that virtual edges correspond to separation pairs of \mathcal{H}' , and that \mathcal{H}' contains no adjacent separation pairs, virtual edges of R may be thought of as paths (in \mathcal{H}') of length at least two. As a consequence, we may ignore such *virtual* separating triangles. Similarly,

regarding Property (I3), when picking the next edge to collapse in R , we do not need to worry about 4-inhibitors that have virtual edges. Formally, this is realized by proving a generalized version of the theorem, in which each edge of \mathcal{H} is labeled as either real or virtual and in which the Properties (I1)–(I4) are relaxed accordingly.

The relaxed versions of Property (I1) and (I2) are easy to achieve. Regarding Property (I4), the main challenge is to ensure that the separation pairs already present in \mathcal{H}' do not become adjacent when performing further edge collapses. Let p, q be a separation pair of \mathcal{H}' . By Observation 12, we may assume that p is the vertex created by collapsing e . Since the *real* separating triangles of \mathcal{H}' are pairwise vertex-disjoint, we conclude that in order for p and q to become adjacent, there must exist a path psq in \mathcal{H}' , where sq is incident to a triangle T of \mathcal{S} . Let R be the rigid triconnected component of \mathcal{H}' that contains T . We compute an edge set \mathcal{K}_R for R that satisfies Properties (I1)–(I4). We show that if sq is constrained by a 4-inhibitor in \mathcal{H} , we may simply obtain \mathcal{K}_R by induction as the Properties (I1)–(I4) of \mathcal{K}_R already suffice to guarantee that p, q do not become adjacent. If sq is not constrained by a 4-inhibitor, there are choices of \mathcal{K}_R for which p, q become adjacent even though Properties (I1)–(I4) hold for \mathcal{K}_R . We show that this is only possible in a very constrained special case, where

- either R has constant size (in this case we select \mathcal{K}_R explicitly, rather than inductively);
- or we can apply a strategy ϱ to replace e with a new edge $\varrho(e)$, so that ϱ is acyclic, that is, $\varrho^i(e) \neq e$, for all $i \in \mathbb{N}$.

Finally, we show that Property (I4) implies Property (I3). To maintain the triconnected components, we use a decremental data structure by Holm et al. [15, Theorem 11] that allows to dynamically maintain an SPQR-tree under a sequence of edge contractions or deletions in $O(n \log^2 n)$ total time. Note that an edge collapse can be implemented using edge contractions and deletions. The runtime of our algorithm is dominated by the replacement strategy ϱ . We show that a single replacement step can be performed in constant time. After at most a linear number of replacements, we obtain an edge that can safely be collapsed. Hence, as the number of collapses is linear, the overall complexity is quadratic. ◀

6 Stellation

Let $K \subset E(G)$ be a set of edges to collapse as described in Theorem 14, and let G' denote the graph that results from simultaneously collapsing the edges from K in G . Then by Property (I3) of Theorem 14 the graph G' does not contain any separating triangle. Let G'' denote the graph that results from stellating all faces in G' , that is, for every nontriangular face f of G' we insert a new vertex v_f into f and we add an edge between v_f and each vertex on ∂f . As discussed in Section 5.2, the following lemma is an easy consequence of Property (I4).

► **Lemma 15.** *The graph G'' does not contain any separating triangle.*

Therefore, we can apply Theorem 4 to G'' to obtain a Hamiltonian cycle C'' for G'' . It remains to address the case that one or two edges of the outer face T_\circ of G are prescribed.

► **Observation 16.** *If any edge of T_\circ is prescribed, then T_\circ is also the outer face of G'' .*

Proof. By Property (P3) we know that T_\circ is also the outer face of G' . By Property (P2) we know that T_\circ a triangle and, therefore, it is not subdivided when going from G' to G'' . ◀

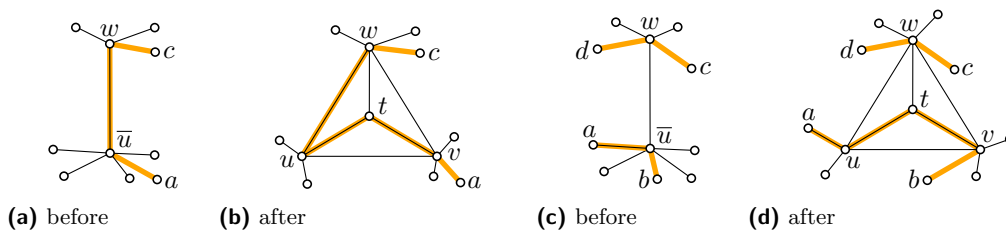
By Observation 16 we can pass any possibly prescribed edge of T_\circ to Theorem 4 so that the obtained Hamiltonian cycle C'' for G'' passes through the(se) prescribed edge(s).

7 Reconstructing Collapses

As a final step to prove Theorem 1, it remains to undo the edge collapses, that is, to go back from the modified graph G'' to the original graph G . Our algorithm maintains a subhamiltonian plane cycle in the current graph, starting with a subhamiltonian plane cycle in G'' . Then the separating triangles of G are processed in a certain order, which is incrementally computed as part of the algorithm. At every step of the algorithm we handle one separating triangle and include its vertices into the current working cycle. For some steps we may choose this separating triangle freely among the remaining ones, while in other steps that choice might be dictated by the previous step.

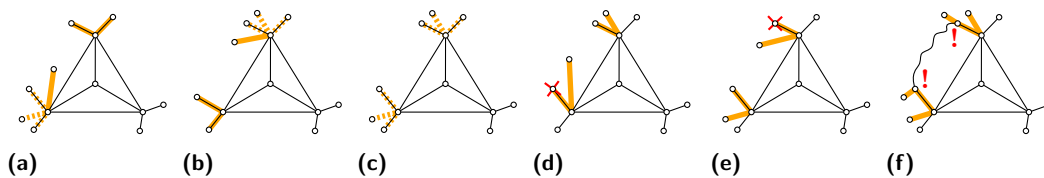
During the whole reconstruction process, we modify the current cycle in specific ways only. In particular, we only modify edges of the cycle that are incident to a separating triangle of G (including vertices that result from the collapse of an edge in K). By Observation 16 and (P3) this assertion suffices to ensure that prescribed edges on the outer face of G (if any) are part of the cycle that is constructed. Our algorithm proceeds in up to three phases.

Phase 1. First, we reconstruct *some* collapsed triangles, while maintaining a subhamiltonian plane cycle in the current graph. We consider only triangles that are visited by the current working cycle in a specific way so that after reconstructing them the cycle can easily be extended to visit all vertices of and in the interior of those triangles; see Figure 5 for examples, where the solid orange segments depict edges of the current cycle.



■ **Figure 5** Examples of two easy reconstructions in Phase 1 (where uv was collapsed to \bar{u}).

At the end of the first phase, we move on to the graph G , that is, we reconstruct all remaining triangles at once. The previously Hamiltonian cycle then becomes a nonspanning cycle in a plane augmentation of G that visits all vertices of G but a pair of vertices from each of the *remaining* triangles (that have not been reconstructed during the first phase already). We can classify these remaining triangles – up to symmetry – into five different types according to how they interact with the current cycle. This classification is illustrated in Figure 6(a)–(e), where the dotted orange segments depict three different options for an edge of the current cycle, and the red crosses mark vertices that are not part of a remaining separating triangle.



■ **Figure 6** The six types of remaining triangles during Phase 2 of the reconstruction.

Phase 2 and 3. During the second phase we then maintain this classification: although a triangle may change its type, it always remains one of these five types. As in the first phase, we process the triangles one by one. Processing a triangle amounts to extending the current cycle to visit the two missing vertices. At the end of Phase 2 we either have a subhamiltonian plane cycle for G (and are done), or we are in a situation where *all* remaining triangles to be handled are of a very specific type with respect to the current cycle, which is illustrated in Figure 6f; the vertices labeled with a red exclamation mark are also part of a remaining separating triangle. Note that this type is a specialization of the more general type depicted in Figure 6c. The remaining triangles – if any – are then processed during a third phase, at the end of which we obtain the desired subhamiltonian plane cycle for G .

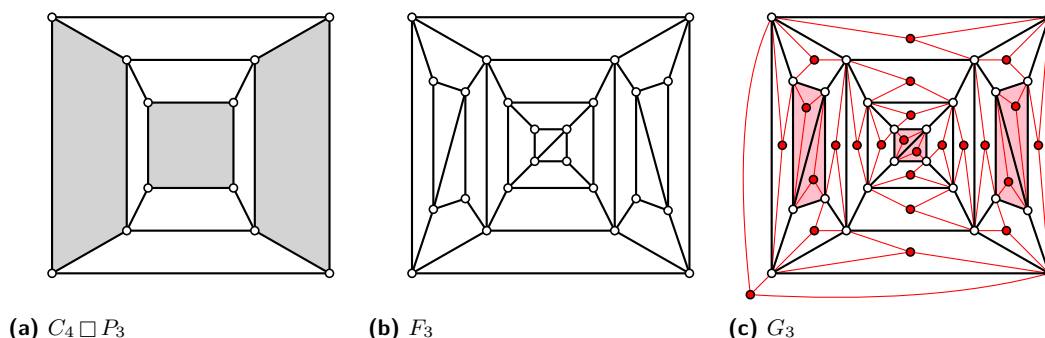
Remarks. The complexity of the reconstruction algorithm is linear overall. To illustrate the challenges for this reconstruction process and why it is important to control the interaction of the remaining triangles with the cycle under construction, consider the examples shown in Figure 7. If the cycle was to visit a remaining triangle in one of the ways depicted, then there is no way to locally modify the cycle to visit the missing vertices. Note that due to the stellation, these two configurations are avoided in the beginning of Phase 1.

8 Conclusions

The graph G_3 whose construction is depicted in Figure 8 is a member of the infinity family described in Theorem 3, which shows that our degree restriction in Theorem 1 is necessary in general. The most prominent open question is whether all planar graphs of degree ≤ 6 are subhamiltonian.



■ **Figure 7** Two types of remaining triangles to avoid: There is no easy way to extend the cycle.



■ **Figure 8** The construction of G_3 , a 3-connected planar graph that is not subhamiltonian and where every vertex of a separating triangle has degree at most six. We start from the Cartesian product $C_4 \square P_3$, where we pick three pairwise nonadjacent faces (shaded in (a)). Then we plant a rectangular prism on each picked face, obtaining the frame F_3 (b). Finally, to obtain G_3 we add a new vertex in every face of F_3 and connect it to three vertices on the boundary (c). The separating triangles of G_3 are shaded red; their vertices have degree six. The red vertices form an independent set, and no edge between any two red vertices can be added while maintaining planarity. As there are 25 red vertices and 24 black vertices, no plane augmentation of G_3 is Hamiltonian.

References

- 1 Armen S. Asratian and Nikolay K. Khachatryan. Some localization theorems on Hamiltonian circuits. *J. Combin. Theory Ser. B*, 49(2):287–294, 1990. doi:10.1016/0095-8956(90)90032-U.
- 2 Frank Bauernöppel. Degree Bounds and Subhamiltonian Cycles in Planar Graphs. *J. Inf. Process. Cybern.*, 23(10–11):529–536, 1987.
- 3 Michael A. Bekos, Martin Gronemann, and Chrysanthi N. Raftopoulou. Two-page book embeddings of 4-planar graphs. *Algorithmica*, 75(1):158–185, 2016. doi:10.1007/s00453-015-0016-8.
- 4 Frank Bernhart and Paul C. Kainen. The book thickness of a graph. *J. Combin. Theory Ser. B*, 27(3):320–331, 1979. doi:10.1016/0095-8956(79)90021-2.
- 5 Therese C. Biedl, Goos Kant, and Michael Kaufmann. On triangulating planar graphs under the four-connectivity constraint. *Algorithmica*, 19(4):427–446, 1997. doi:10.1007/PL00009182.
- 6 Jean Cardinal, Michael Hoffmann, Vincent Kusters, Csaba D. Tóth, and Manuel Wettstein. Arc diagrams, flip distances, and Hamiltonian triangulations. *Comput. Geom. Theory Appl.*, 68:206–225, 2018. doi:10.1016/j.comgeo.2017.06.001.
- 7 Norishige Chiba and Takao Nishizeki. The Hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs. *J. Algorithms*, 10(2):187–211, 1989. doi:10.1016/0196-6774(89)90012-6.
- 8 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. doi:10.1007/978-3-540-77974-2.
- 9 Gabriel A. Dirac. Some theorems on abstract graphs. *Proc. London Math. Soc.*, s3-2(1):69–81, 1952. doi:10.1112/plms/s3-2.1.69.
- 10 Günter Ewald. Hamiltonian Circuits in Simplicial Complexes. *Geom. Dedicata*, 2:115–125, 1973. doi:10.1007/BF00149287.
- 11 Michael R. Garey, David S. Johnson, and Robert E. Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM J. Comput.*, 5(4):704–714, 1976. doi:10.1137/0205049.
- 12 Xiaxia Guan and Weihua Yang. Embedding 5-planar graphs in three pages. *CoRR*, abs/1801.07097, 2018. arXiv:1801.07097.
- 13 Carsten Gutwenger and Petra Mutzel. A linear time implementation of SPQR-trees. In *Proc. 8th Internat. Sympos. Graph Drawing (GD'00)*, volume 1984 of *LNCS*, pages 77–90. Springer, 2000. doi:10.1007/3-540-44541-2_8.
- 14 Lenwood S. Heath. *Algorithms for embedding graphs in books*. Phd thesis, University of North Carolina, Chapel Hill, 1985. URL: <http://www.cs.unc.edu/techreports/85-028.pdf>.
- 15 Jacob Holm, Giuseppe F. Italiano, Adam Karczmarz, Jakub Łącki, and Eva Rotenberg. Incremental SPQR-trees for planar graphs. In *Proc. 26th Annu. European Sympos. Algorithms (ESA 2018)*, volume 112 of *LIPICs*, pages 46:1–46:16, 2018. doi:10.4230/LIPICs.ESA.2018.46.
- 16 Øystein Ore. Note on Hamilton circuits. *Amer. Math. Monthly*, 67(1):55, 1960. doi:10.2307/2308928.
- 17 Daniel P. Sanders. On paths in planar graphs. *J. Graph Theory*, 24(4):341–345, 1997. doi:10.1002/(SICI)1097-0118(199704)24:4<341::AID-JGT6>3.0.CO;2-0.
- 18 Andreas Schmid and Jens M. Schmidt. Computing Tutte paths. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *LIPICs*, pages 98:1–98:14, 2018. doi:10.4230/LIPICs.ICALP.2018.98.
- 19 William T. Tutte. A theorem on planar graphs. *Trans. Amer. Math. Soc.*, 82(1):99–116, 1956. doi:10.1090/S0002-9947-1956-0081471-8.
- 20 Avi Wigderson. The complexity of the Hamiltonian circuit problem for maximal planar graphs. Technical Report 298, Princeton University, 1982. URL: <http://www.math.ias.edu/~avi/PUBLICATIONS/MYPAPERS/W82a/tech298.pdf>.

58:14 Triconnected Planar Graphs of Maximum Degree Five are Subhamiltonian

- 21 Mihalis Yannakakis. Four pages are necessary and sufficient for planar graphs (extended abstract). In *Proc. 18th Annu. ACM Sympos. Theory Comput. (STOC 1986)*, pages 104–108, 1986. doi:10.1145/12130.12141.
- 22 Mihalis Yannakakis. Embedding planar graphs in four pages. *J. Comput. Syst. Sci.*, 38(1):36–67, 1989. doi:10.1016/0022-0000(89)90032-9.

Parallel Weighted Random Sampling

Lorenz Hübschle-Schneider

Karlsruhe Institute of Technology, Germany
huebschle@kit.edu

Peter Sanders

Karlsruhe Institute of Technology, Germany
sanders@kit.edu

Abstract

Data structures for efficient sampling from a set of weighted items are an important building block of many applications. However, few parallel solutions are known. We close many of these gaps both for shared-memory and distributed-memory machines. We give efficient, fast, and practicable algorithms for sampling single items, k items with/without replacement, permutations, subsets, and reservoirs. We also give improved sequential algorithms for alias table construction and for sampling with replacement. Experiments on shared-memory parallel machines with up to 158 threads show near linear speedups both for construction and queries.

2012 ACM Subject Classification Theory of computation → Sketching and sampling; Theory of computation → Parallel algorithms; Theory of computation → Data structures design and analysis

Keywords and phrases categorical distribution, multinoulli distribution, parallel algorithm, alias method, PRAM, communication efficient algorithm, subset sampling, reservoir sampling

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.59

Related Version A full version of the paper is available at <https://arxiv.org/abs/1903.00227>.

Supplement Material The code and scripts used for our experiments are available under the GPLv3 at <https://github.com/lorenzhs/wrs>.

1 Introduction

Weighted random sampling asks for sampling items (elements) from a set such that the probability of sampling item i is proportional to a given weight w_i . Several variants of this fundamental computational task appear in a wide range of applications in statistics and computer science, *e.g.*, for computer simulations, data analysis, database systems, and online ad auctions (see, *e.g.*, Motwani et al. [26], Olken et al. [28]). Continually growing data volumes (“Big Data”) imply that the input sets and even the sample itself can become large. Since actually processing the sample is often fast, sampling algorithms can easily become a performance bottleneck. Due to the hardware developments of the last years, this means that we need *parallel algorithms* for weighted sampling. This includes *shared-memory* algorithms that exploit current multi-core processors, and *distributed algorithms* that split the work across multiple machines without incurring too much overhead for communication.

However, there has been surprisingly little work on parallel weighted sampling. This paper closes many of these gaps. Table 1 summarizes our results on the following widely used variants of the weighted sampling problem. We process the input set $A = 1..n$ on p processing elements (PEs) where $i..j$ is a shorthand for $\{i, \dots, j\}$. Item i has weight w_i and $W := \sum_{i=1}^n w_i$. Define $u := \log U$ where $U := w_{\max}/w_{\min} := \max_i w_i / \min_i w_i$.



© Lorenz Hübschle-Schneider and Peter Sanders;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 59; pp. 59:1–59:24



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

WRS-1: Weighted sampling of *one item* from a categorical (or multinoulli) distribution (equivalent to WRS-R and WRS-N for $k = 1$).

WRS-R: Sample k items from A *with replacement*, *i.e.*, the samples are independent and for each sample X , $\mathbf{P}[X = i] = w_i/W$. Let $s = |S| \leq k$ denote the number of *different* items in the sample S . Note that we may have $s \ll k$ for skewed input distributions.

WRS-N: Sample k pairwise unequal items $s_1 \neq \dots \neq s_k$ *without replacement* such that $\mathbf{P}[s_l = i] = w_i/(W - w_{s_1} - \dots - w_{s_{l-1}})$.

WRP: Permute the elements with the same process as for WRS-N using $k = n$.

WRS-S: Sample a *subset* $S \subseteq A$ where $\mathbf{P}[i \in S] = w_i \leq 1$.

WRS-B: *Batched reservoir sampling.* Repeatedly solve WRS-N when batches of new items arrive. Only the current sample and batch may be stored. Let b denote the batch size.

When applicable, our algorithms build a data structure once which is later used to support fast sampling queries. Most of the algorithms have linear work and variants with logarithmic (or even constant) latency. Neither competitive parallel algorithms nor more efficient sequential algorithms are known. The distributed algorithms are refinements of the shared-memory algorithms with the goal to reduce communication costs compared to a direct distributed implementation of the shared-memory algorithms. As a consequence, each PE mostly works on its local data (the owner-computes approach). Communication – if at all – is only performed to coordinate the PEs and is sublinear in the local work except for extreme corner cases. The owner-computes approach introduces the complication that differences in local work introduce additional parameters into the analysis that characterize the local work in different situations (*e.g.*, the last line of Table 1). The summary in Table 1 therefore covers the case when items are randomly assigned to PEs. This simplifies the exposition and is actually an approach that one can sometimes take in practice.

Outline

First, in Section 2, we review the models of computation used in this paper as well as known techniques we are building on. We discuss additional related work in Section 3. In Section 4, we consider Problem WRS-1. We first give an improved sequential algorithm for constructing

■ **Table 1** Result overview (expected and asymptotic). Distributed results assume random distribution of inputs. Input size n , output size s , sample size k , startup latency of point-to-point communication α , time for communicating one machine word β , log-weight ratio $u = \log U = \log w_{\max}/w_{\min}$, mini-batches of b items per PE. The complexity of sorting n integers with keys from $0..x$ is $\text{isort}_x(n)$ (isort^* = parallel, isort^1 = sequential).

Problem	Shared Memory					Distributed		
	§	Preprocessing		Query		§	Preprocessing	Query
		Work	Span	Work	Span		Time	Time
WRS-1	4.2	n	$\log n$	1	1	4.3	$\frac{n}{p} + \alpha \log p$	α
WRS-R	5	$\text{isort}_u^*(n)$	$s + \log n$	$s + \log n$	$\log n$	5.1	$\text{isort}_u^1(\frac{n}{p}) + \alpha \log p$	$\frac{s}{p} + \log p$
WRS-N	6	$\text{isort}_u^*(n)$	$k + \log n$	$k + \log n$	$\log n$	6	$\text{isort}_u^1(\frac{n}{p}) + \alpha \log p$	$\frac{k}{p} + \alpha \log^2 n$
WRS-N						6	$\frac{n}{p} + \beta u + \alpha \log p$	$\frac{k}{p} + \alpha \log n$
WRP	7	—	—	$\text{isort}_{n(u+\log n)}^*(n)$		7	—	$\text{isort}_{n(u+\log n)}^*(n)$
WRS-S	8	n	$\log n$	$s + \log n$	$\log n$	8	$\frac{n}{p} + \log p$	$\frac{s}{p} + \log p$
WRS-B	—	—	—	—	—	9	—	$b \log(b+k) + \alpha \log^2 kp$

alias tables – the most widely used data structure for Problem WRS-1 that allows sampling in constant time. Then we parallelize that algorithm for shared and distributed memory. We also present parallel construction for a more space efficient variant.

Sampling k items with replacement (Problem WRS-R) seems to be trivially parallelizable with an alias table. However this does not lead to a communication-efficient distributed algorithm and we can generally do better for skewed input distributions where the number of *distinct* output elements s can be much smaller than k . Section 5 develops such an algorithm which is interesting both as a parallel and a sequential algorithm.

Section 6 employs the algorithm for Problem WRS-R to solve Problem WRS-N. The main difficulty here is to estimate the right number of samples with replacement to obtain a sufficient number of distinct samples. Then an algorithm for WRS-N without preprocessing is used to reduce the “weighted oversample” to the desired exact output size.

It is well known that the weighted permutation Problem WRP can be reduced to sorting (see Section 2.3). We show in Section 7 that this is actually possible with linear work by appropriately defining the (random) sorting keys so that we can use integer sorting with a small number of different keys. Since previous linear-time algorithms are fairly complicated [20], this may also be interesting for a sequential algorithm. Indeed, a similar approach might also work for other problems where sorting can be a bottleneck, *e.g.*, smoothed analysis of approximate weighting matching [24].

For subset sampling (Problem WRS-S), we parallelize the approach of Bringmann et al. [6] in Section 8. Once more, the preprocessing requires integer sorting. However, only $\mathcal{O}(\log n)$ different keys are needed so that linear work sorting works with logarithmic latency even deterministically on a CREW PRAM.

In Section 9, we adapt the sequential streaming algorithm of Efrimidis et al. [12] to a distributed setting where items are processed in small batches. This can be done in a communication efficient way using our previous work on distributed priority queues [16].

Section 10 gives a detailed experimental evaluation of our algorithms for WRS-1 and WRS-R. Section 11 summarizes the results and discusses possible future directions.

2 Preliminaries

2.1 Models of Computation

We strive to describe our parallel algorithms in a model-agnostic way, *i.e.*, we largely describe them in terms of standard operations such as prefix sums for which efficient parallel algorithms are known on various models of computation. We analyze the algorithms for two simple models of computation. In each case p denotes the number of processing elements (PEs). Most of our algorithms achieve polylogarithmic running time for a sufficiently large number of PEs. This is a classical goal in parallel algorithm theory and we believe that it is now becoming practically important with the advent of massively parallel (“exascale”) computing and fine-grained parallelism in GPGPU.

For shared-memory algorithms we use the CREW PRAM model (concurrent read exclusive write parallel random access machine) [18]. We will use the concepts of total *work* and *span* of a computation to analyze these algorithms. The span of a computation is the time needed by a parallel algorithm with an unbounded number of PEs.

For distributed-memory computations we use point-to-point communication between PEs where exchanging a message of length ℓ takes time $\alpha + \ell\beta$. We assume $1 \leq \beta \leq \alpha$. We will use that *prefix sums* and (*all-reductions*) can be computed in time $\mathcal{O}(\beta\ell + \alpha \log p)$ for vectors of size ℓ . The *all-gather* operation collects a value from each PE and delivers all values to

all PEs. It can be implemented to run in time $\mathcal{O}(\beta p + \alpha \log p)$ [19]. We will particularly strive to obtain *communication-efficient algorithms* [35] where total communication cost is sublinear in the local computations. Some of our algorithms are even *communication free*.

We need one basic toolbox operation where the concrete machine model has some impact on the complexity. Sorting n items with integer keys from $1..K$ can be done with linear work in many relevant cases. Sequentially, this is possible if K is polynomial in n (radix sort). Radix sort can be parallelized even on a distributed-memory machine with linear work and span n^ε for any constant $\varepsilon > 0$. Logarithmic span is possible for $K = \mathcal{O}(\log^c n)$ for any constant c , even on an EREW PRAM [30, Lemma 3.1]. For a CRCW PRAM, expected linear work and logarithmic span can be achieved when $K = \mathcal{O}(n \log^c n)$ [30] (the paper gives the constraint $K = \mathcal{O}(n)$) but the generalization is obvious and important for us in Section 7). Resorting to comparison based algorithms, we get work $\mathcal{O}(n \log n)$ and $\mathcal{O}(\log n)$ span on an EREW PRAM [8].

2.2 Bucket-Based Sampling

The basic idea behind several solutions of Problem WRS-1 is to build a table of $m = \Theta(n)$ buckets where each bucket represents a total weight of W/m . Sampling then selects a random bucket uniformly at random and uses the information stored in the bucket to determine the actual item. If item weights differ only by a constant factor, we can simply store one item per bucket and use rejection sampling to obtain constant expected query time (see, e.g., Devroye [9], Olken et al. [28]).

Deterministic sampling with only a single memory probe is possible using Walker's alias table method [38], and its improved construction due to Vose [37]. An alias table consists of $m := n$ buckets where bucket $b[i]$ represents some part w'_i of the weight of item i . The remaining weight of the heavier items is distributed to the remaining capacity of the buckets such that each bucket only represents one other item (the *alias* a_i). Algorithm 1 gives high-level pseudocode for the approach proposed by Vose. The items are first classified into light and heavy items. Then the heavy items are distributed over light items until their residual weight drops below W/n . They are then treated in the same way as light items.

To sample an item, pick a bucket index r uniformly at random, toss a biased coin that comes up heads with probability $w'_r n/W$, and return r for heads, or $b[r].a$ for tails.

■ **Algorithm 1** Classical construction of alias tables similar to Vose's approach [37].

```

Procedure voseAliasTable( $\langle w_1, \dots, w_n \rangle$ ,  $b$  : Array of  $w$  :  $\mathbb{R} \times a$  :  $\mathbb{N}$ )
   $W := \sum_i w_i$  -- total weight
   $h := \{i \in 1..n : w_i > W/n\}$  : Stack -- heavy items
   $\ell := \{i \in 1..n : w_i \leq W/n\}$  : Stack -- light items
  for  $i := 1$  to  $n$  do  $b[i].w := w_i$  -- init buckets with weights
  while  $h \neq \emptyset$  do -- consume heavy items
     $j := h.\text{pop}$  -- get a heavy item
    while  $b[j].w > W/n$  do -- still heavy
       $i := \ell.\text{pop}$  -- get a light item
       $b[i].a := j$  -- Fill bucket  $b[i]$  with a ...
       $b[j].w := (b[j].w + b[i].w) - W/n$  -- ... piece of item  $j$ .
     $\ell.\text{push}(j)$  -- Bucket  $j$  is light now.

```

2.3 Weighted Sampling using Exponential Variates

It is well known that an unweighted sample without replacement of size k out of n items $1..n$ can be obtained by associating with each item a uniform variate $v_i := \mathbf{rand}()$, and selecting the k with the smallest associated variates. This method can be generalized to generate a *weighted* sample without replacement by raising uniform variates to the power $1/w_i$ and selecting the k items with the *largest* associated values [11, 12, 10]. Equivalently, one can generate exponential random variates $v_i := -\ln(\mathbf{rand}())/w_i$ and select the k items with the *smallest* associated v_i [2] (“*exponential clocks method*”), which is numerically more stable.

2.4 Divide-and-Conquer Sampling

Uniform sampling with and without replacement can be done using a divide-and-conquer algorithm [34]. To sample k out of n items uniformly and with replacement, split the set into two subsets with n' (left) and $n - n'$ (right) items, respectively. Then the number of items k' to be sampled from the left has a binomial distribution (k trials with success probability n'/n). We can generate k' accordingly and then recursively sample k' items from the left and $k - k'$ items from the right. This can be used for a communication-free parallel sampling algorithm. We have a tree with p leaves. Each leaf represents a subproblem of size about $n/p - 1$ for each PE. Each PE descends this tree to the leaf assigned to it (time $\mathcal{O}(\log p)$) and then generates the resulting number of samples (time $\mathcal{O}(k/p + \log p)$ with high probability). Different PEs have to draw the same random variates for the same interior node of the tree. This can be achieved by seeding a pseudo-random number generator with an ID of this node.

3 Related Work

3.1 Sampling one Item (Problem WRS–1)

Extensive work has been done on generating discrete random variates from a fixed distribution [38, 37, 21, 9, 6]. All these approaches use preprocessing to construct a data structure that subsequently supports very fast (constant time) sampling of a single item. Bringmann et al. [5] explain how to achieve expected time r using only $\mathcal{O}(n/r)$ bits of space beyond the input distribution itself. There are also dynamic versions that allow efficient weight updates. Some (rather complicated ones) allow that even in constant expected time [15, 22].

3.2 Sampling Without Replacement (Problems WRS–N and WRP)

The exponential clocks method of Section 2.3 is an $\mathcal{O}(n)$ algorithm for sampling without replacement. This approach also lends itself towards use in streaming settings (*reservoir sampling*) and can be combined with a skip value distribution to reduce the number of required random variates from $\mathcal{O}(n)$ to $\mathcal{O}(k \log \frac{n}{k})$ [12]. A related algorithm for WRS–N with given inclusion probabilities instead of relative weights is described by Chao [7].

More efficient algorithms for WRS–N repeatedly sample an item and remove it from the distribution using a dynamic data structure [40, 28, 15, 22]. With the most efficient such algorithms [15, 22] we achieve time $\mathcal{O}(k)$, albeit at the price of an inherently sequential and rather complicated algorithm that might have considerable hidden constant factors.

It is also possible to combine techniques for sampling *with* replacement with a rejection method. However, the performance of these methods depends heavily on U , the ratio between the largest and smallest weight in the input, as the rejection probability rises steeply once the heaviest items are removed. Lang [20] gives an analysis and experimental evaluation of such methods for the case of $k = n$ (*cf.* “Permutation” below). A recent practical evaluation of approaches that lend themselves towards efficient implementation is due to Müller [27].

3.3 Parallel Sampling

There is surprisingly little work on parallel sampling. Even uniform unweighted sampling had many loose ends until recently [34]. Parallel uniformly random permutations are covered in [14, 33]. Efraimidis and Spirakis note that WRS-N can be solved in parallel with span $\mathcal{O}(\log n)$ and work $\mathcal{O}(n \log n)$ [11]. They also note that solving the selection problem suffices if the output need not be sorted. The optimal dynamic data structure for WRS-1 [22] admits a parallel bulk update in the (somewhat exotic) combining-CRCW-PRAM model. However, this does not help with Problem WRS-N since batch sizes are one.

4 Alias Table Construction (Problem WRS-1)

4.1 Improved Sequential Alias Tables

Before discussing parallel alias table construction, we discuss a simpler, faster and more space efficient sequential algorithm that is a better basis for parallelization. Previous algorithms need auxiliary arrays/queues of size $\Theta(n)$ in order to decide in which order the buckets are filled. Vose [37] mentions that this can be avoided but does not give a concrete algorithm. We now describe an algorithm with this property.

The idea of the algorithm is that two indices i and j sweep the input array with respect to light and heavy items, respectively. The loop invariant is that the weight of items corresponding to light (heavy) items preceding i (j) has already been distributed over some buckets and that their corresponding buckets have already been constructed. Variable w stores the weight of the part of item j that has not yet been assigned to buckets. Each iteration of the main loop advances one of the indices and initializes one bucket. When the residual weight w exceeds W/n , item j is used to fill bucket i , the residual weight w is reduced by $W/n - w_i$, and index i is advanced to the next light item. Otherwise, the remaining weight of heavy item j fits into bucket j and the remaining capacity of bucket j is filled with the next heavy item. Algorithm 2 gives pseudocode that emphasizes the high degree of symmetry between these two cases.

■ **Algorithm 2** A sweeping algorithm for building alias tables.

```

Procedure sweepingAliasTable( $\langle w_1, \dots, w_n \rangle$ ,  $b$  : Array of  $w : \mathbb{R} \times a : \mathbb{N}$ )
   $W := \sum_i w_i$  -- total weight
   $i := \min \{k > 0 : w_k \leq W/n\}$  -- first light item
   $j := \min \{k > 0 : w_k > W/n\}$  -- first heavy item
   $w := w_j$  -- residual weight of current heavy item
  while  $j \leq n$  do
    if  $w > W/n$  then -- Pack a light bucket.
       $b[i].w := w_i$  -- Item  $i$  completely fits here.
       $b[i].a := j$  -- Item  $j$  fills the remainder of bucket  $i$ .
       $w := (w + w_i) - W/n$  -- Update residual weight of item  $j$ .
       $i := \min \{k > i : w_k \leq W/n\}$  -- next light item, assume  $w_{n+2} = 0$ 
    else -- Pack a heavy bucket.
       $b[j].w := w$  -- Now item  $j$  completely fits here.
       $b[j].a := j' := \min \{k > j : w_k > W/n\}$  -- next heavy item, assume  $w_{n+1} = \infty$ 
       $w := (w + w_{j'}) - W/n$  -- Find residual weight of item  $j'$ .
       $j := j'$ 

```

4.2 Parallel Alias Tables

The basic idea behind our *splitting based algorithm* is to identify subsets L and H of light ($w_i \leq W/n$) and heavy ($w_i > W/n$) items such that they can be allocated precisely within their respective buckets, *i.e.*, $w(H \cup L) := \sum_{i \in H \cup L} w_i = (|H| + |L|) \cdot W/n$. By splitting the items into such pairs of subsets, we can perform alias table construction for these subsets in parallel. Since the above balance condition cannot always be achieved, we allow to “steal” a piece of a further heavy item, *i.e.*, this item can be used to fill buckets in several subproblems. Such a split item will only be used as an alias except in the last subproblem where it is used. Thus, the computed data structure is still an alias table.

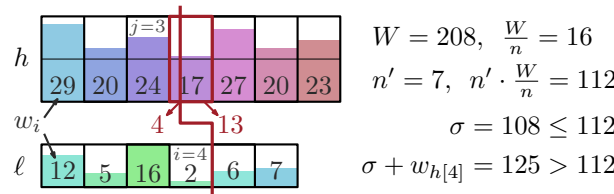
We first explain how to split n items into two subsets of size n' and $n - n'$. Similar to Vose’s algorithm, we first compute arrays ℓ and h containing the indices of the light and heavy items, respectively. We then determine indices i and j such that $i + j = n'$, $\sigma := \sum_{x \leq i} w_{\ell[x]} + \sum_{x \leq j} w_{h[x]} \leq n'W/n$ and $\sigma + w_{h[j+1]} > n'W/n$; see Figure 1 for an example. These values can be determined by binary search over the value of j . By precomputing prefix sums of the weights of the items in ℓ and h , each iteration of the binary search takes constant time. The resulting subproblem then consists of the light items $L := \{\ell[1], \dots, \ell[i]\}$, the heavy items $H := \{h[1], \dots, h[j]\}$ and a fraction of size $n'W/n - \sigma$ of item $h[j + 1]$.

To split the input into p independent subproblems of near-equal size, we perform the above two-way-split for the values $n'_k = \lceil nk/p \rceil$ for $k \in 1..p - 1$. PE k is then responsible for filling a set of buckets corresponding to sets of light and heavy items, each represented by a range of indices into ℓ and h . A piece of a further heavy item may be used to make the calculation work out. Note that a subproblem might contain an empty set of light or heavy items and that a single heavy item j may be assigned partially to multiple subproblems, but only the last PE using a heavy item will fill its bucket.

Algorithm 3 gives detailed pseudocode. It uses function `split` to compute $p - 1$ different splits in parallel. The result triple (i, j, s) of `split` specifies that buckets $\ell[1] \dots \ell[i]$ as well as $h[1] \dots h[j]$ shall be filled using the left subproblem. Moreover, total weight s of item $h[j + 1]$ is *not used* on the left side, *i.e.*, spilled over to the right side.

This splitting information is then used to make p parallel calls to procedure `pack` – giving each PE the task to fill up to $\lceil n/p \rceil$ buckets. `Pack` has input parameters specifying ranges of heavy and light items it should use. The parameter `spill` determines how much weight of item $h[j - 1]$ can be used for that. `Pack` works similar to the sweeping algorithm from Algorithm 2. If the residual weight of item $h[j - 1]$ drops below W/n , this item is actually also packed in this call. The body of the main loop is dominated by one if-then-else case distinction. When the residual weight of the current heavy item falls below W/n , its bucket is filled using the next heavy item. Otherwise, its weight is used to fill the current light item.

► **Theorem 1.** *We can construct an alias table with work $\Theta(n)$ and span $\Theta(\log n)$ on a CREW PRAM.*



■ **Figure 1** Parallel alias tables: splitting $n = 13$ items into two parts of size $n' = 7$ and $n - n' = 6$.

■ **Algorithm 3** Pseudocode for parallel splitting based alias table construction (PSA).

```

Procedure psaAliasTable( $\langle w_1, \dots, w_n \rangle$ ,  $b$  : Array of  $w : \mathbb{R} \times a : \mathbb{N}$ )
   $W := \sum_i w_i$  -- total weight
   $h := \{i \in 1..n : w_i > W/n\}$  : Array -- parallel traversal finds heavy items and
   $\ell := \{i \in 1..n : w_i \leq W/n\}$  : Array -- and light items
  for  $k := 1$  to  $p - 1$  dopar  $(i_k, j_k, \text{spill}_k) := \text{split}(\lceil nk/p \rceil)$  -- split into  $p$  pieces
   $(i_0, j_0, \text{spill}_0) := (0, 0, 0)$ ;  $(i_p, j_p) := (n, n)$  -- cover corner cases
  for  $k := 1$  to  $p$  dopar  $\text{pack}(i_{k-1} + 1, i_k, j_{k-1} + 1, j_k, \text{spill}_{k-1})$ 

Function split( $n'$ ) :  $\mathbb{N} \times \mathbb{N} \times \mathbb{R}$ 
   $a := 1$ ;  $b := \min(n', |h|)$  --  $a..b$  is search range for  $j$ 
  loop -- binary search
     $j := \lfloor (a + b)/2 \rfloor$  -- bisect search range
     $i := n' - j$  -- Establish the invariant  $i + j = n'$ .
     $\sigma := \sum_{x \leq i} w_{\ell[x]} + \sum_{x \leq j} w_{h[x]}$  -- work to the left; use precomputed prefix sums
    if  $\sigma \leq n'W/n$  and  $\sigma + w_{h[j+1]} > n'W/n$  then return  $(i, j, w_{h[j+1]} + \sigma - n'W/n)$ 
    if  $\sigma \leq n'W/n$  then  $b := j - 1$  else  $a := j + 1$  -- narrow search range

  (* pack buckets  $b[\ell[\underline{i}]], \dots, b[\ell[\bar{i}]]$  and buckets  $b[h[\underline{j}]], \dots, b[h[\bar{j}]]$ . *)
  (* Use up to weight spill from item  $h[\underline{j} - 1]$ . *)
  Procedure pack( $\underline{i}, \bar{i}, \underline{j}, \bar{j}, \text{spill}$ )
     $i := \underline{i}$  --  $\ell[i]$  is the current light item.
     $j := \underline{j} - 1$  --  $h[j]$  is the current heavy item.
     $w := \text{spill}$  -- part of current heavy item still to be placed
    if  $\text{spill} = 0$  then  $j++$ ;  $w := w_{h[j]}$ 
    loop
      if  $w \leq W/n$  then -- pack a heavy bucket
        if  $j > \bar{j}$  then return
         $b[h[j]].w := w$ 
         $b[h[j]].a := h[j + 1]$ 
         $w := (w + w_{h[j+1]}) - W/n$ 
         $j++$ 
      else -- pack a light bucket
        if  $i > \bar{i}$  then return
         $b[\ell[i]].w := w_{\ell[i]}$ 
         $b[\ell[i]].a := h[j]$ 
         $w := (w + w_{\ell[i]}) - W/n$ 
         $i++$ 

```

Proof. The algorithm requires linear work and logarithmic span for identifying light and heavy items and for computing prefix sums [4] over them. Splitting works in logarithmic time. Then each PE needs time $\mathcal{O}(n/p)$ to fill the buckets assigned to its subproblem. ◀

4.3 Distributed Alias Table Construction

The parallel algorithm described in Section 4.2 can also be adapted to a distributed-memory machine. However, this requires information about all items to be communicated. Hence, more communication efficient algorithms are important for large n . To remedy this problem, we will now view sampling as a 2-level process implementing the owner-computes approach underlying many distributed algorithms.

Let E_i denote the set of items allocated to PE i . For each PE i , we create a *meta-item* of weight $W_i := \sum_{j \in E_i} w_j$. Sampling now amounts to sampling a meta-item and then delegating the task to sample an actual item from E_i to PE i . The local data structures can be built independently on each PE.¹ In addition, we need to build a data structure for sampling a meta-item. There are several variants in this respect with different trade-offs:

► **Theorem 2.** *Assuming that $\mathcal{O}(n/p)$ elements are allocated to each PE, we can sample a single item in time $\mathcal{O}(\alpha)$ after preprocessing a 2-level alias table, which can be done in time $\mathcal{O}(n/p)$ plus the following communication overhead*

$$\beta p + \alpha \log p \text{ with replicated preprocessing} \tag{1}$$

$$\alpha \log^2 p \text{ expected time using the algorithm from Section 4.2} \tag{2}$$

$$\alpha \log p \text{ with only expected time bounds for the query} \tag{3}$$

Proof. Building the local alias tables takes time $\mathcal{O}(\max_i |E_i|) = \mathcal{O}(n/p)$ sequentially. For Equation (1), we can perform an all-gather operation on the meta-items and compute the data structure for the meta-items in a replicated way.

For Equation (2) and Equation (3), we can compute an alias table for the meta-items using a parallel algorithm. Sampling then needs an additional indirection. First, a meta-bucket j is computed. Then a request is sent to PE j which identifies the subset E_i from which the item should be selected and delegates that task of sampling from E_i to PE i .² Equation (2) then follows by using the shared-memory algorithm from Section 4.2. It can be implemented to run in expected time $\mathcal{O}(\alpha \log^2 p)$ on a distributed-memory machine using PRAM emulation [31].

At the price of getting only expected query time, we can also achieve logarithmic latency (Equation (3)) by using the rejection sampling algorithm from Section 4.4. The preprocessing there requires only prefix sums that can directly be implemented on distributed memory: We have to assign p meta-items to $2p$ meta-buckets (two on each PE). Suppose PE i computes the prefix sum $k = \sum_{j < i} \lceil W_j/W \rceil$. It then sends item i to PE $j = \lfloor k/2 \rfloor$. PE j then initiates a broadcast of item i to PEs $j.. \lfloor (j + \lfloor W_i/W \rfloor - 1)/2 \rfloor$. All of this is possible in time $\mathcal{O}(\alpha \log p)$. ◀

4.3.1 Redistributing Items

As discussed so far, *constructing* distributed-memory 2-level alias tables is communication efficient. However, when large items are predominantly allocated on few PEs, *sampling* many items can lead to an overload on PEs with large W_i . We can remedy this problem by moving large items to different PEs or even by splitting them between multiple PEs. This redistribution can be done in the same way we construct alias tables. This implies a trade-off between redistribution cost (part of preprocessing) and load balance during sampling.

We now look at the case where an adversary can choose an arbitrarily skewed distribution of item sizes but where the items are randomly allocated to PEs (or that we actively randomize the allocation implying $\mathcal{O}(n/p)$ additional communication volume).

¹ Possibly using a shared-memory parallel algorithm locally.

² If we ensure that meta-items have similar size (see Section 4.3.1) then we can arrange the meta-items in such a way that $i = j$ most of the time.

► **Theorem 3.** *If items are randomly distributed over the PEs initially, it suffices to redistribute $\mathcal{O}(\log p)$ items from each PE such that afterwards each PE has total weight $\mathcal{O}(W/p)$ in expectation and $\mathcal{O}(n/p + \log p)$ (pieces of) items. This redistribution takes expected time $\mathcal{O}(\alpha \log^2 p)$ when supporting deterministic queries (Theorem 2-(2)) and expected time $\mathcal{O}(\alpha \log p)$ using rejection sampling (Theorem 2-(3)).*

Proof. Let us distinguish between *heavy* items that are larger than $cW/(p \log p)$ for an appropriate constant c and the remaining *light* items. The expected maximum weight allocated to a PE based on light items is $\mathcal{O}(W/p)$ [32].

There can be at most $p \log(p)/c$ heavy items. By standard balls into bins arguments, only $\mathcal{O}(\log p)$ heavy items can initially be allocated to any PE with high probability. We use the algorithm from Theorem 2-(2) to distribute the heavy items to p meta-buckets of remaining capacity $\max(0, W/p - S_i)$ where S_i is the total weight of the light items allocated to PE i . Using the bound from Equation (2) would result in a time bound of $\mathcal{O}(\log^3 p)$ since we have a factor $\mathcal{O}(\log p)$ more items. However, the only place where we need a full-fledged PRAM emulation is for doing the binary search which takes only $\mathcal{O}(\log p)$ steps on the PRAM and time $\mathcal{O}(\alpha \log^2 p)$ when emulated on distributed memory.

For the faster variant with rejection sampling, we use prefix sums to distribute the largest $N := \mathcal{O}(p \log p)$ items such that each PE gets an even share of it. For this, we build groups of $N/p = \mathcal{O}(\log p)$ items that we distribute in an analogous fashion to the proof in Theorem 2-(3) – a prefix sum, followed by forwarding a group followed by a segmented broadcast. The asymptotic complexity does not change since even messages of size $\mathcal{O}(\log p)$ can be broadcast in time $\mathcal{O}(\alpha \log p)$, *e.g.*, using pipelining. Finally, each PE unpacks the group it received and extracts the parts that it has to represent in the meta-table. ◀

4.4 Compressed Data Structures for WRS–1

Bringmann and Larsen [5] give a construction similar to alias tables that allows expected query time $\mathcal{O}(r)$ using $2n/r + o(n)$ bits of additional space. We describe the variant for $r = 1$ in some more detail. We assign $\lceil w_i/W \rceil$ buckets to each item, *i.e.*, $\leq 2n$ in total. Item i is assigned to buckets $\sum_{j < i} \lceil w_j/W \rceil .. \sum_{j \leq i} \lceil w_j/W \rceil - 1$. A query samples a bucket j uniformly at random. Suppose bucket j is assigned to item i . If $j \in \sum_{j < i} \lceil w_j/W \rceil .. \sum_{j \leq i} \lceil w_j/W \rceil - 2$, item i is returned. If $j = \sum_{j \leq i} \lceil w_k/W \rceil - 1$, item i is returned with probability $\lceil w_i/W \rceil - \lfloor w_i/W \rfloor$. Otherwise, bucket j is rejected and the query starts over. Since the overall success probability is $\geq 1/2$, the expected query time is constant.

The central observation for compression is that it suffices to store one bit for each bucket that indicates whether a new item starts at bucket $b[i]$. When querying bucket j , the item stored in it can be determined by counting the number of 1-bits up to position j . This *rank*-operation can be supported in constant time using an additional data structure with $o(n)$ bits. Further reduction in space is possible by representing r items together as one entry in b .

Both constructing the bit vector and constructing the rank data structure is easy to parallelize using prefix sums (for adding scaled weights and counting bits, respectively) and embarrassingly parallel computations. Shun [36] even gives a bit parallel algorithm needing only $\mathcal{O}(n/\log n)$ work for computing the rank data structure. We get the following result:

► **Theorem 4.** *Bringmann and Larsen’s $n/r + o(n)$ bit data structure can be built using $\mathcal{O}(n)$ work and $\mathcal{O}(\log n)$ span allowing queries in expected time $\mathcal{O}(r)$.*

5 Output Sensitive Sampling With Replacement (Problem WRS–R)

The algorithm of Section 4.2 easily generalizes to sampling k items with replacement by simply executing k queries. Since the precomputed data structures are immutable, these queries can be run in parallel. We obtain optimal span $\mathcal{O}(1)$ and work $\mathcal{O}(k)$.

► **Corollary 5.** *After a suitable alias table data structure has been computed, we can sample k items with replacement with work $\mathcal{O}(k)$ and span $\mathcal{O}(1)$.*

Yet if the weights are skewed this may not be optimal since large items will be sampled multiple times. Here, we describe an *output sensitive* algorithm that outputs only different items in the sample together with how often they were sampled, *i.e.*, a set S of pairs (i, k_i) indicating that item i was sampled k_i times. The work will be proportional to the output size s up to a small additive term.

Note that outputting multiplicities may be important for appropriately processing the samples. For example, let X denote a random variable where item i is sampled with probability w_i/W and suppose we want a truthful estimator for the expectation of $f(X)$ for some function f . Then $\sum_{(i,k_i) \in S} k_i f(i)/k$ is such an estimator.

We will combine and adapt three previously used techniques for related problems: the bucket tables from Section 2.2, the divide-and-conquer technique from Section 2.4 [34], and the subset sampling algorithm of Bringmann et al. [6].

We approximately sort the items into $u = \lceil \log U \rceil$ groups of items whose weights differ by at most a factor of two – weight w_i is mapped to group $\lceil \log(w_i/w_{\min}) \rceil$.

To help determine the number of samples to be drawn from each group, we build a complete binary tree with one *nonempty* group at each leaf. Interior nodes store the total weight of items in groups assigned to their subtrees. This *divide-and-conquer tree* (*DC-tree*) allows us to generalize the technique from Section 2.4 to weighted items. Suppose we want to sample k elements from a subtree rooted at an interior node whose left subtree has total weight L and whose right subtree has total weight R . Then the number of items k' to be sampled from the left has a binomial distribution (k trials with success probability $L/(L+R)$). We can generate k' accordingly and then recursively sample k' items from the left subtree and $k - k'$ items from the right subtree. A recursive algorithm can thus split the number of items to be sampled at the root into numbers of items sampled at each group. When a subtree receives no samples, the recursion can be stopped. Since the distribution of weights to groups can be highly skewed, this stopping rule will be important in the analysis.

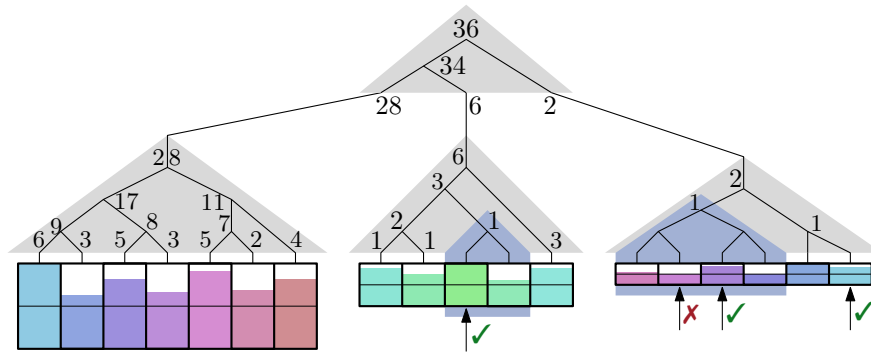
For each group G , we integrate bucket tables and DC-tree as follows. For the bucket table we can use a very simple variant that stores n_G items with weights from the interval $[a, 2a)$ in n_G buckets of capacity $2a$. Sampling one element then uses a rejection method that repeats the sampling attempt when the random variate leads to an empty part of a bucket.³

We also build a DC-tree for each group. A simple linear mapping of items into the bucket table allows us to associate a range of relevant buckets b_T with each subtree T of the DC-tree.

For sampling m items from a group G , we use the DC-tree to decide which subtree has to contribute how many samples. When this number decreases to 1 for a subtree T , we sample this element directly and in constant expected time from the buckets in the range b_T .

Figure 2 gives an example. We obtain the following complexities:

³ If desired, we can also avoid rejection sampling by mapping the items without gaps into up to $2n_G$ buckets of size a . This way there are at most two items in each bucket. Note that this is still different from alias tables because we need to map ranges of consecutive items to ranges of buckets. This is not possible for alias tables.



■ **Figure 2** Output-sensitive sampling: assignment of multiplicities with $k = 36$.

► **Theorem 6.** *Preprocessing for Problem WRS-R can be done in the time and span needed for integer sorting n elements with $u = \lceil \log U \rceil = \lceil \log(w_{\max}/w_{\min}) \rceil$ different keys⁴ plus linear work and logarithmic span (on a CREW PRAM) for building the actual data structure. Using this data structure, sampling a multiset S with k items and s different items can be done with span $\mathcal{O}(\log n)$ and expected work $\mathcal{O}(s + \log n)$ on a CREW PRAM.*

Proof. Besides sorting the items into groups, we have to build binary trees of total size $\mathcal{O}(n)$. This can be done with logarithmic span and linear work using a simple bottom-up reduction. The bucket-tables which have total size n can be constructed as in Section 4.2.

The span of a query is essentially the depth of the trees, $\log u + \log n \leq 2 \log n$.

Bounding the work for a query is more complicated since, in the worst case, the algorithm can traverse paths of logarithmic length in the DC-trees for just a constant number of samples taken at its leaves. However, this is unlikely to happen and we show that in expectation the overhead is a constant factor plus an additive logarithmic term. We are thus allowed to charge a constant amount of work to each different item in the output and can afford a leftover logarithmic term.

We first consider the top-most DC-tree T that divides samples between groups. Tree T is naturally split into a *heavy* range of groups that contain some items which are sampled with probability at least $1/2$ and a remaining range of *light* groups in which all items are sampled with probability at most $1/2$. Assuming the heavy groups are to the left, consider the path P in T leading to the first light group. Subtrees branching from P to the left are complete subtrees that lead to heavy groups only. Since all leaves represent non-empty groups, we can charge the cost for traversing the left trees to the elements in the groups at the leaves – in expectation, at least half of these groups contain elements that are actually sampled.

Then follow at most $2 \log n$ light groups that have a probability $\geq 1/n$ to yield at least one sample. These middle groups fit into subtrees of T of logarithmic total size and hence cause logarithmic work for traversing them.

The expected work for the remaining very light groups can be bounded by their number ($\leq u \leq n$) times the length of the path in T leading to them ($\leq \log u \leq \log n$) times the probability that they yield at least one sample ($\leq 1/n$). The product ($\leq n \log(n)/n = \log n$) is another logarithmic term.

Finally, Lemma 7 shows that the work for traversing DC-trees within a group is linear in the output size from each group. Summing this over all groups yields the desired bound. ◀

⁴ Section 2.1 discusses the cost of this operation on different models of computation.

► **Lemma 7.** *Consider a DC-tree plus bucket array for sampling with replacement of k out of n items where weights are in the range $[a, 2a)$. Then the expected work for sampling is $\mathcal{O}(s)$ where s is the number of different sampled items.*

Proof. If $k \geq n$, $\Omega(n)$ items are sampled in expectation at a total cost of $\mathcal{O}(n)$. So assume $k < n$ from now on. The first $\log k + \mathcal{O}(1)$ levels of T may be traversed completely, contributing a total cost of $\mathcal{O}(k)$.

For the lower levels, we count the number Y of visited nodes from which at least 2 items are sampled. This is proportional to the total number of visited nodes since nodes from which only one item is sampled contribute only constant expected cost (for directly sampling from the array) and since there are at most $2Y$ such nodes.

Let X denote the number of items sampled at a node at level ℓ of tree T . An interior node at level ℓ represents $2^{L-\ell}$ leaves with total weight $W_\ell \leq 2a2^{L-\ell}$ where $L = \lceil \log n \rceil$. X has a binomial distribution with k trials and success probability

$$\rho = \frac{W_\ell}{W} \leq \frac{2a2^{L-\ell}}{a2^{L-1}} = 4 \cdot 2^{-\ell} .$$

Hence,

$$\mathbf{P}[X \geq 2] = 1 - \mathbf{P}[X = 0] - \mathbf{P}[X = 1] = 1 - (1-p)^k - kp(1-p)^{k-1} \approx (k\rho)^2/2$$

where the “ \approx ” holds for $k\rho \ll 1$ and was obtained by series development in the variable $k\rho$.

The expected cost at level $\ell > \log k + \mathcal{O}(1)$ is thus estimated as

$$2^\ell \mathbf{P}[X \geq 2] \approx 2^\ell (k\rho)^2/2 \leq 2^\ell (k \cdot 4 \cdot 2^{-\ell})^2/2 = 8k^2 2^{-\ell} .$$

At level $\ell = \lceil \log k \rceil + 3 + i$ we thus get expected cost $\leq k2^{-i}$. Summing this over i yields total cost $Y = \mathcal{O}(k)$. ◀

5.1 Distributed Case

The batched character of sampling with replacement makes this setting even more adequate for a distributed implementation using the owner-computes approach. Each PE builds the data structure described above for its local items. Furthermore, we build a top-level DC-tree that distributes the samples between the PEs, *i.e.*, with one leaf for each PE. We will see below that this can be done using a bottom-up reduction over the total item weights on each PE, *i.e.*, no PRAM emulation or replication is needed. Each PE only needs to store the partial sums appearing on the path in the reduction tree leading to its leaf. Sampling itself can then proceed without communication – each PE simply descends its path in the top-level DC-tree analogous to the uniform case [34]. Afterwards, each PE knows how many samples to take from its local items. Note that we assume k to be known on all PEs and that communication for computing results from the sample is not considered here.

► **Theorem 8.** *Sampling k out of n items with replacement (Problem WRS-R) can be done in a communication-free way with processing overhead $\mathcal{O}(\log p)$ in addition to the time needed for taking the local sample. Building and distributing the DC-tree for distributing the samples is possible in time $\mathcal{O}(\alpha \log p)$.*

Proof. It remains to explain how the reduction can be done in such a way that it can be used as a DC-tree during a query and such that each PE knows the path in the reduction tree leading to its leaf. First assume that $p = 2^d$ is a power of two. Then we can use

the well known hypercube algorithm for all-reduce (e.g., [19]). In iteration $i \in 1..d$ of this algorithm, a PE knows the sum for its local $i - 1$ dimensional subcube and receives the sum for the neighboring subcube along dimension i to compute the sum for its local i dimensional subcube. For building the DC-tree, each PE simply records all these values.

For general values of p , we first build the DC tree for $d = \lfloor \log p \rfloor$. Then, each PE i with $i < 2^d$ and $j = i + 2^d < p$ receives the aggregate local item weight from PE j and then sends its path to PE j . ◀

Similar to Section 4.3, it depends on the assignment of the items to the PEs whether this approach is load balanced for the local computations. Before, we needed a balanced distribution of both number of items and item weights. Now the situation is better because items may be sampled multiple times but require work only once. On the other hand, we do not want to split heavy items between multiple PEs since this would increase the amount of work needed to process the sample. It would also undermine the idea of communication-free sampling if we had to collect samples of the same item assigned to different PEs. Below, we once more analyze the situation for items with arbitrary weight that are allocated to the PEs randomly.

► **Theorem 9.** *Consider an arbitrary set of item sizes and let $u = \log(\max_i w_i / \min_i w_i)$. If items are randomly assigned to the PEs initially, then preprocessing takes expected time $\mathcal{O}(\text{isort}_u^1(n/p) + \alpha \log p)$ where $\text{isort}_u^1(x)$ denotes the time for sequential integer sorting of x elements using keys from the range $0..u$.⁵ Using this data structure, sampling a multiset S with k items and s different items can be done in expected time $\mathcal{O}(s/p + \log p)$.*

Proof. For preprocessing, standard Chernoff bound arguments tell us that $\mathcal{O}(n/p + \log p)$ items will be assigned to a PE with high probability. Since sorting is now a local operation, we only need an efficient sequential algorithm for approximately sorting integers. The term $\alpha \log p$ is for the global DC-tree as in Theorem 8.

A sampling operation will sample s items. Since their allocation is independent of the choice of the sampled items, we can once more use Chernoff bounds to conclude that only $\mathcal{O}(s/p + \log p)$ of them are allocated to any PE with high probability. ◀

6 Sampling k Items Without Replacement (Problem WRS-N)

We can construct an algorithm for sampling without replacement based on the output-sensitive algorithm for sampling *with* replacement of Section 5. Presume we know an $\ell > k$ so that a sample of size ℓ with replacement contains at least k and no more than $\mathcal{O}(k)$ unique items. Then we can obtain a sample with $k' \geq k$ different items using the algorithm of Section 5, discard the multiplicities, and downsample to size k using the exponential clocks method (see Section 2.3).

To find the right value for ℓ , we derive an estimation of the number of unique samples as a function of ℓ . The basis of this estimation is to assume that sufficiently heavy items are sampled once and lighter items are sampled with probability proportional to their weight. We precompute the data needed for the estimation for each group and then perform a binary search over the groups. More concretely, when the currently considered group stores elements with weights in the range $[a, 2a)$, we try the value $\ell = \lceil 1/(2a) \rceil$. We (over)estimate the resulting number of unique samples as

$$|\{i : w_i \geq a\}| + \ell \cdot \frac{\sum \{w_i : w_i < a\}}{W} .$$

⁵ Note that this will be linear in all practically relevant situations.

In the full paper we show that this is a good estimate and how to use it to drive the binary search.

7 Permutation (Problem WRP)

As already explained in Section 2.3, weighted permutation can be reduced to sorting random variates of the form $-\ln(r)/w_i$ where r is a uniform random variate. The nice thing is that a lot is known about parallel sorting. The downside is that sorting may need superlinear work in the worst case. However, since we are sorting *random* numbers, we may still get linear expected work. This is well known when sorting uniform random variates; *e.g.*, [25, Theorem 5.9]. The idea is to map the random variates in linear time to a small number of buckets such that the occupancy of a bucket is bounded by a binomial distribution with constant expectation. Then the buckets can be sorted using a comparison based algorithm without getting more than linear work in total.

In the full paper, we explain how to achieve the same for the highly skewed distribution needed for WRP by applying radix sort and the monotonous transformation function $f(r, w_i) := n \ln(-\ln(r)nw_{\max}/w_i)$.

8 Subset Sampling (Problem WRS-S)

Subset sampling is a generalization of Bernoulli Sampling to the weighted case. The unweighted case can be solved in expected time linear in the output size by computing the geometrically distributed distances between elements in the sample [1]. The naïve algorithm for the weighted problem, which consists of throwing a biased coin for each item, requires $\mathcal{O}(n)$ time. Bringmann et al. [6] show that this is optimal if only a single subset is desired, and present a sequential algorithm that is also optimal for multiple queries.

The difference between WRS-S on the one hand and WRS-1/WRS-R on the other hand is that we do not have a fixed sample size but rather treat the item weights as independent inclusion probabilities in the sample (this requires $w_i \leq 1$ for all i). Hence, different algorithms are required. Observe that the expected sample size is $W \leq n$. Then our goal is to devise a parallel preprocessing algorithm with work $\mathcal{O}(n)$ which subsequently permits sampling with work $\mathcal{O}(1 + W)$.

In the full paper we parallelize the approach of Bringmann et al. [6]. Similar to our algorithm for sorting with replacement, this algorithm is based on grouping items into sets with similar weight. In each group, one can use ordinary Bernoulli sampling in connection with rejection sampling. Load balanced division between PEs can be done with a prefix sum calculation over the weights in each group.

9 Sampling with a Reservoir

In the full paper, we adapt the streaming algorithm of Efrimidis et al. [12] to a distributed mini-batch streaming model, where PEs process variable-size batches of items one at a time. The PEs' memory is too small to store previous batches, only the current mini-batch is available in memory. This is a generalization of the traditional data stream model and widely used in practice, *e.g.*, in Apache Spark Streaming [41], where it is called *discretized streams*. The basic idea is to keep the reservoir in a distributed priority queue [16].

10 Experiments

We now report experiments on alias tables (Problem WRS-1, Section 4) and the closely related problem of sampling with replacement (Problem WRS-R, Section 5).

Experimental Setup. We use machines with Intel and AMD processors in our experiments. The Intel machine has four Xeon Gold 6138 CPUs (4×20 cores, 160 hyper-threads, of which we use up to 158 to minimize the influence of system tasks on our measurements) and 768 GiB of DDR4-2666 main memory. The AMD machine is a single-socket system with a 32-core AMD EPYC 7551P CPU (64 hyper-threads, of which we use up to 62) and 256 GiB of DDR4-2666 RAM. While single-socket, this machine also has non-uniform memory access (NUMA), as the CPU consists of four dies internally. Both machines run Ubuntu 18.04. All implementations are in C++ and compiled with GNU g++ 8.2.0 (flags `-O3 -fno`).

Our measurements do not include time spent on memory allocation and mapping.

Implementation Details. We implemented alias table construction using our parallel splitting algorithm (PSA, Section 4.2) and output-sensitive sampling with replacement (OS, Section 5), as well as a shared-memory version of the distributed algorithm (2vl, Section 4.3, Theorem 2-(1)). The 2vl algorithm can either use Vose’s method (2vl-classic, Algorithm 1) or our sweeping algorithm of Section 4.1 (2vl-sweep, Algorithm 2) as base case. For OS, we use an additional optimization that aborts the tree descent and uses the base case bucket table when fewer than 128 samples are to be drawn from at least half as many items. The resulting elements are then deduplicated using a hash table to ensure that each element occurs only once in the output. A variant without this deduplication is called OS-ND and may be interesting if items *may* be returned multiple times. We also implemented sequential versions of both alias table construction methods. Both of the machines used require some degree of Non-Uniform Memory Access (NUMA) awareness in memory-bound applications like ours. Thus, in our parallel implementations, all large arrays are distributed over the available NUMA nodes, and threads are pinned to NUMA nodes to maintain data locality. All pseudorandom numbers are generated with 64-bit Mersenne Twisters [23], using the Intel Math Kernel Library (MKL) [17] on the Intel machine and dSFMT⁶ on the AMD machine. All of our implementations are publicly available under the GNU General Public License (version 3) at <https://github.com/lorenzhs/wrs>.

Compared to the descriptions in Sections 2.2 and 4.2, we performed a minor modification to construction of the tables to improve query performance. In the alias table, we store tuples $(w_i, A = [i, a_i])$ of a weight w_i , item i and alias a_i . This allows for an optimization at query time, where we return $A[\text{rand}() \cdot W/n \geq w_i]$, saving a conditional branch. When indices are 32-bit integers and weights are 64-bit doubles, this does not use additional space since the record size is padded to 128 bits anyway.

Sequential Performance. Surprisingly, many common existing implementations of alias tables (*e.g.*, `gsl_rand_discrete_preproc` in the GNU Scientific Library (GSL) [13] or `sample` in the R project for statistical computing [29]) use a struct-of-arrays memory layout for the alias table data structure. By using the array-of-structs paradigm instead, we can greatly improve memory locality, incurring one instead of up to two cache misses per query. Combined with branchless selection inside the buckets and a faster random number generator,

⁶ <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/>, version 2.2.3

our query is more than three times as fast as that of GSL version 2.5 (measured for $n = 10^8$). At the same time, alias table construction using our implementation of Vose’s method is 30 % faster than GSL. Other popular statistics packages, such as NumPy (version 1.5.1, function `np.choice`) or Octave (Statistics package version 1.4.0, function `randsample`) employ algorithms with superlinear query time. We therefore use our own implementation of Vose’s algorithm as the baseline in our evaluation.

Among our sequential implementations, construction with Vose’s method is slightly faster than our sweeping algorithm. On the Intel machine, it is 8 % faster, while on the AMD machine, the gap is 3 %. However, since all of our measurements exclude the time for memory allocations, this is not the full picture. If we include memory allocation, our method is around 5 % faster than Vose’s on both machines. This is because it requires no additional space, compared to $\mathcal{O}(n)$ auxiliary space for Vose’s method.

The optimization described above to make queries branchless lowers query time substantially, namely by 22 % on the Intel machine and 27 % on the AMD machine, again for $n = 10^8$. Storing the item indices at construction time comes at no measurable extra cost.

10.1 Construction

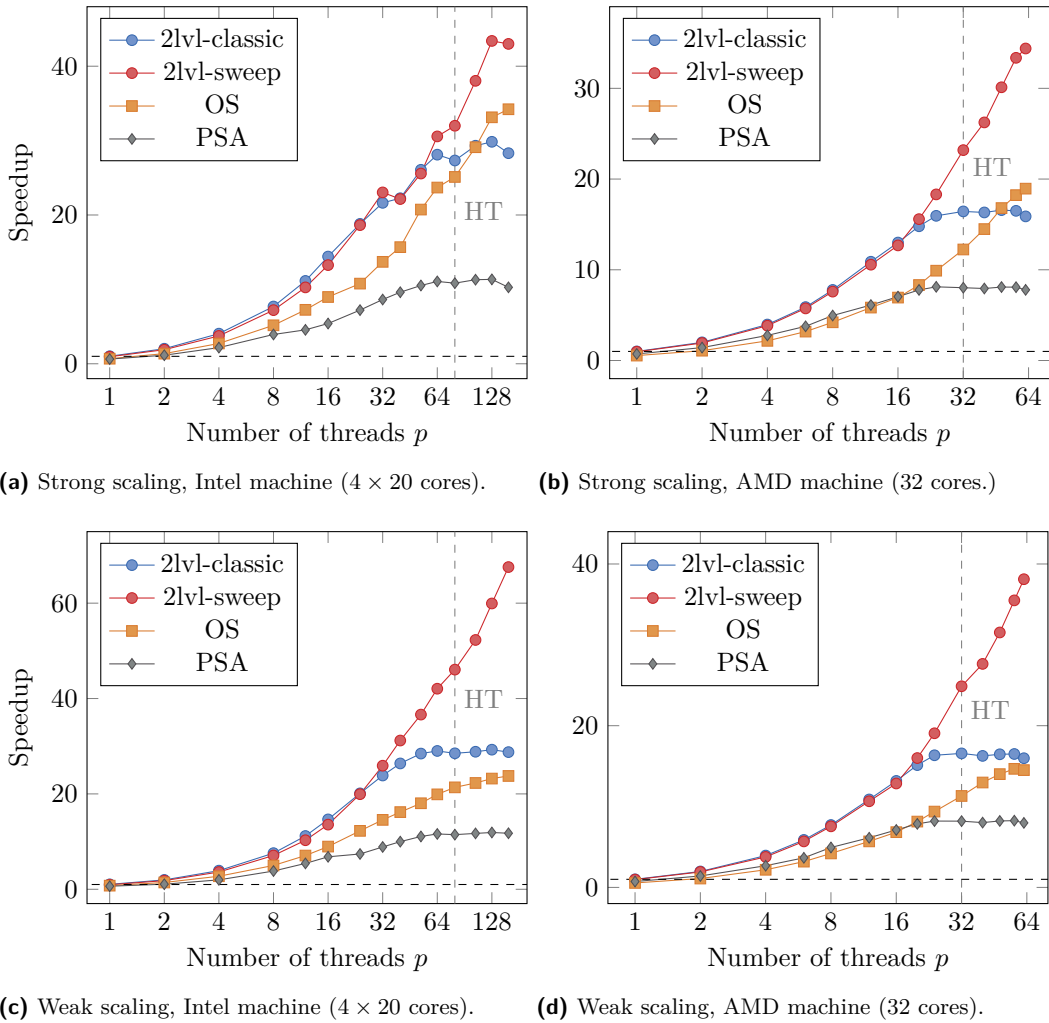
Speedups compared to an optimized sequential implementation of Vose’s alias table construction algorithm are shown in Figure 3 (strong scaling with $n = 10^8$ and weak scaling with $n/p = 10^7$ uniform random variates). Speedups do not increase further once the machine’s memory bandwidth is saturated, limiting the speedup that can be achieved with techniques that require multiple passes over the data (PSA, 2lvl-classic). In contrast, 2lvl-sweep can be constructed almost independently by the PEs and requires much fewer accesses to memory. Sequentially, there is little difference between our sweeping algorithm and Vose’s method. However, our algorithm scales much better to high thread counts because it reduces the memory traffic and since hyper-threading (HT) helps to hide the overhead of branch mispredictions. This is especially visible on the AMD machine (Figure 3b), where 2lvl-sweep achieves more than twice the speedup of 2lvl-classic, 34 compared to 16. The lack of scaling for 2lvl-classic and 2lvl-sweep when going from 32 to 40 cores on the Intel machine (Figure 3a) coincides with a large frequency reduction in the CPUs at this point [39]. Preprocessing for OS introduces some overhead but is not much slower than 2lvl.

In the weak scaling experiments (Figures 3c and 3d), we again see clearly how 2lvl-classic and PSA are limited by memory bandwidth. Using more than two threads per available memory channel (4×6 for the Intel machine, 8 for the AMD machine) yields nearly no additional benefit for these algorithms. Meanwhile, 2lvl-sweep and OS are not limited by the available memory bandwidth, but rather latency of memory accesses. As a result, they scale well even to the highest thread counts.

10.2 Queries

We performed strong and weak scaling experiments for queries (Figure 4) as well as throughput measurements for different sample sizes (Figure 5). Besides uniform random variates, we use random permutations of the weights $\{1^{-s}, 2^{-s}, \dots, n^{-s}\}$ for a parameter s to obtain more skewed “power-law” distributions.

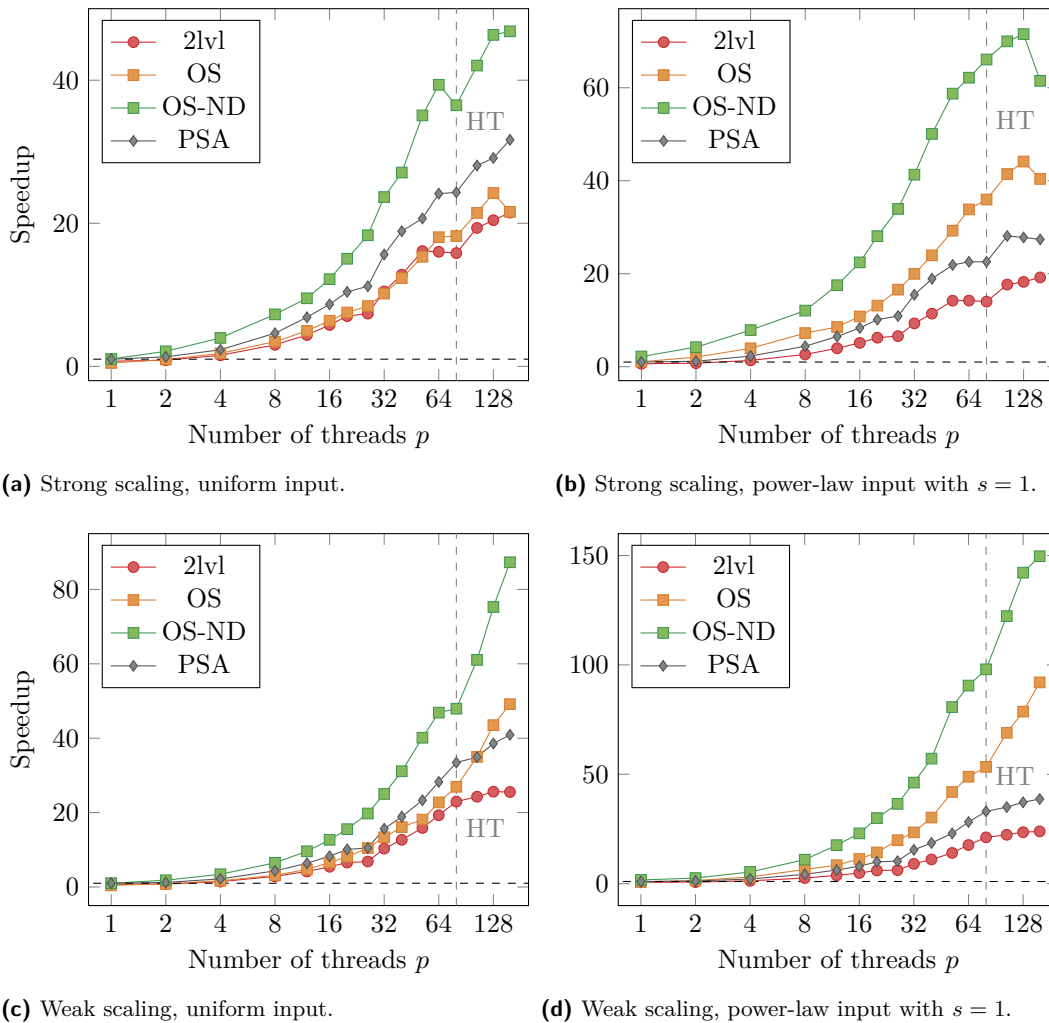
Scaling. First, consider the scaling experiments of Figure 4. These experiments were conducted on the Intel machine, as its highly non-uniform memory access characteristics highlight the differences between the algorithms. All speedups are given relative to sampling



■ **Figure 3** Strong (top) and weak (bottom) scaling evaluation of parallel alias table construction techniques. Strong scaling with input size $n = 10^8$, weak scaling with $n/p = 10^7$. Speedups are measured relative to our optimized implementation of Vose’s method (Algorithm 1, Section 2.2).

sequentially from an alias table. The strong scaling experiments (Figures 4a and 4b) deliberately use a small sample size to show scaling to low per-thread sample counts ($\approx 64\,000$ for 158 threads). We can see that all algorithms have good scaling behavior. Hyper-threading (marked “HT” in the plots) yields additional speedups, as it helps to hide memory latency. This already shows that sampling is bound by random access latency to memory. Sampling from PSA and 2vl is done completely independently by all threads, with no interaction apart from reading from the same shared data structures. Because the Intel machine has four NUMA zones, most queries have to access another NUMA node’s memory. This limits the speedups achievable using PSA and 2vl.

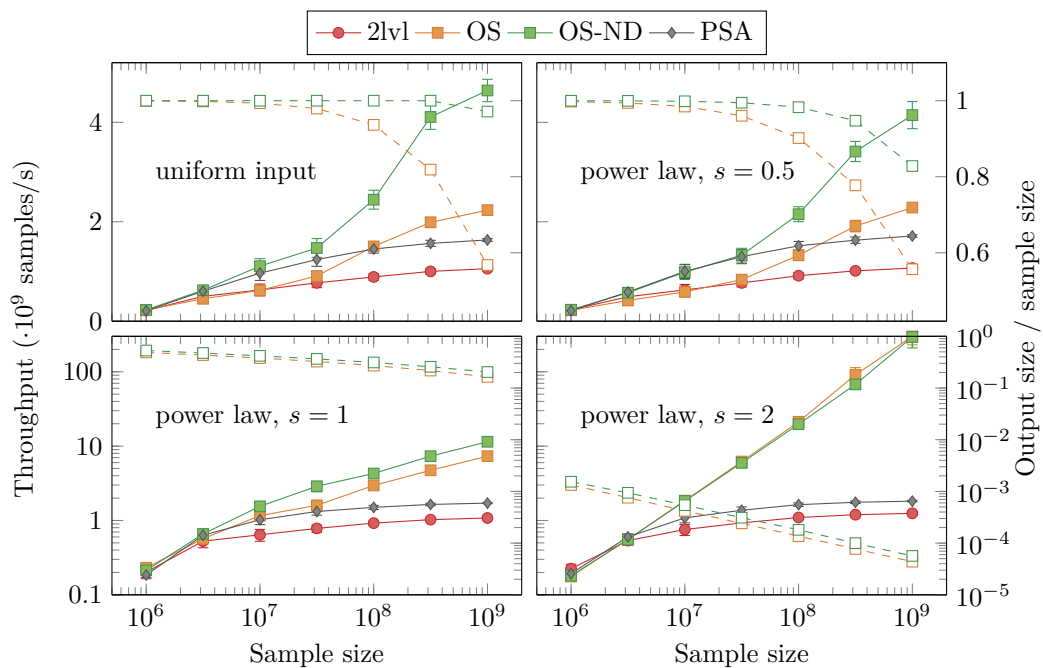
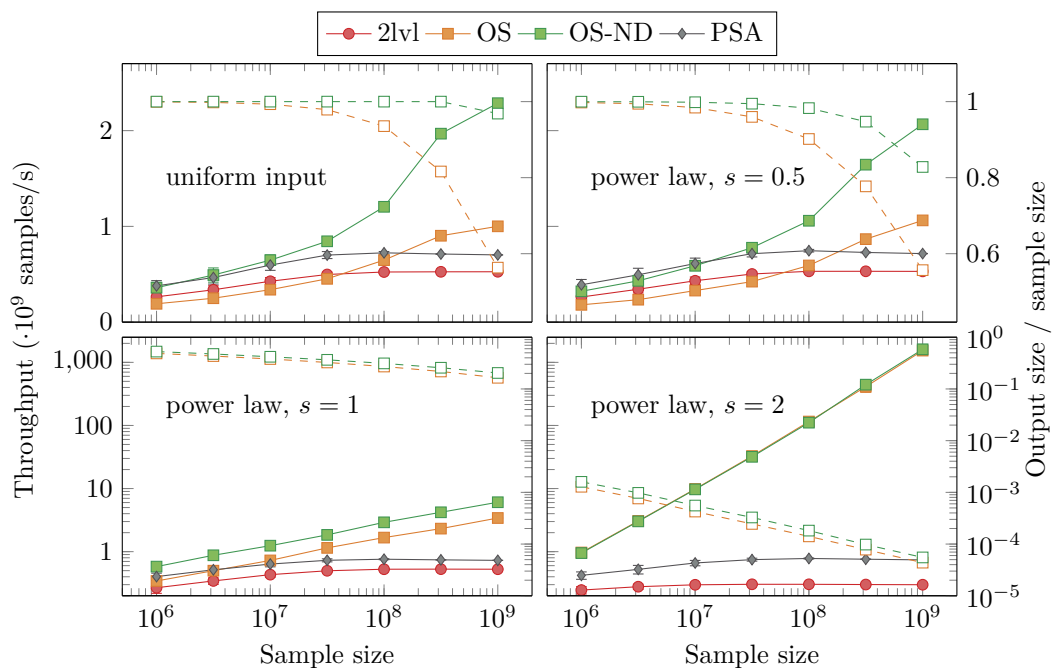
On the other hand, OS and OS-ND have a shared top-level sample assignment stage, after which threads only access local memory. This benefits scaling, especially on NUMA machines. As a result, OS-ND achieves the best speedups, despite this benchmark producing very few samples with multiplicity greater than one (Figure 4a, sample size is 1% of input size). On the other hand, deduplication in the base case of OS has significant overhead, making it roughly 25% slower than sampling from an alias table for such inputs, even sequentially.



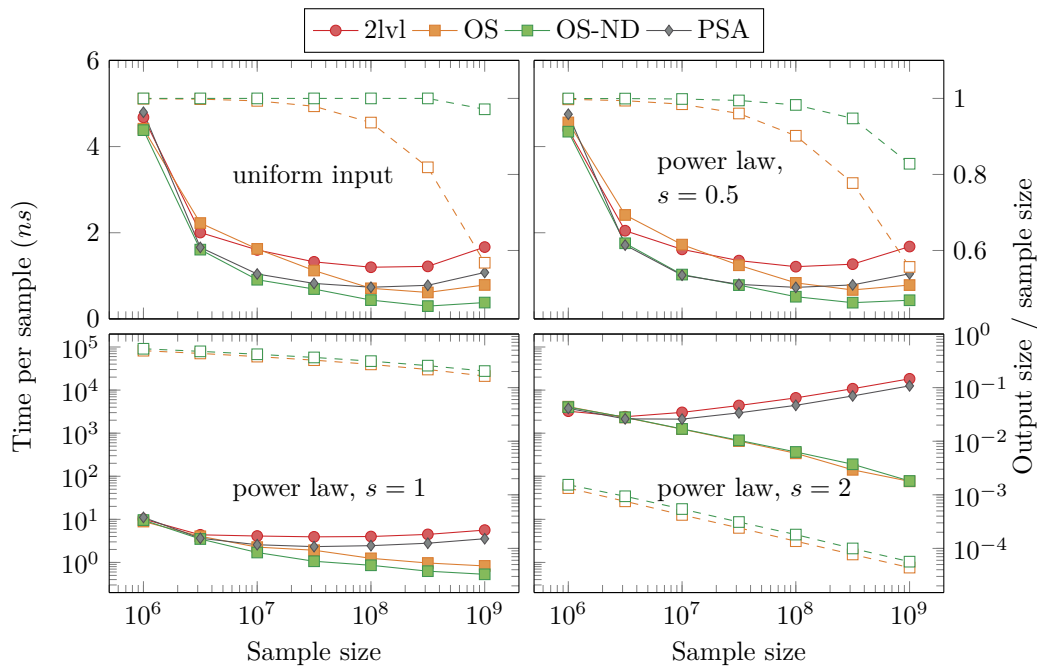
■ **Figure 4** Query strong and weak scaling for $n = 10^9$ input elements. Sample size for strong scaling $s = 10^7$, per-thread sample size for weak scaling $s/p = 10^6$. All speedups relative to sequential alias tables. Intel machine.

The weak scaling experiments of Figures 4c and 4d show even better speedups because many more samples are drawn here than in our strong scaling experiment, reducing overheads. Sampling from a classical alias table (PSA) achieves a speedup of 40 here, again limited by memory accesses rather than computation. Meanwhile, the output-sensitive methods (OS, OS-ND) reap the benefits of accessing only local memory.

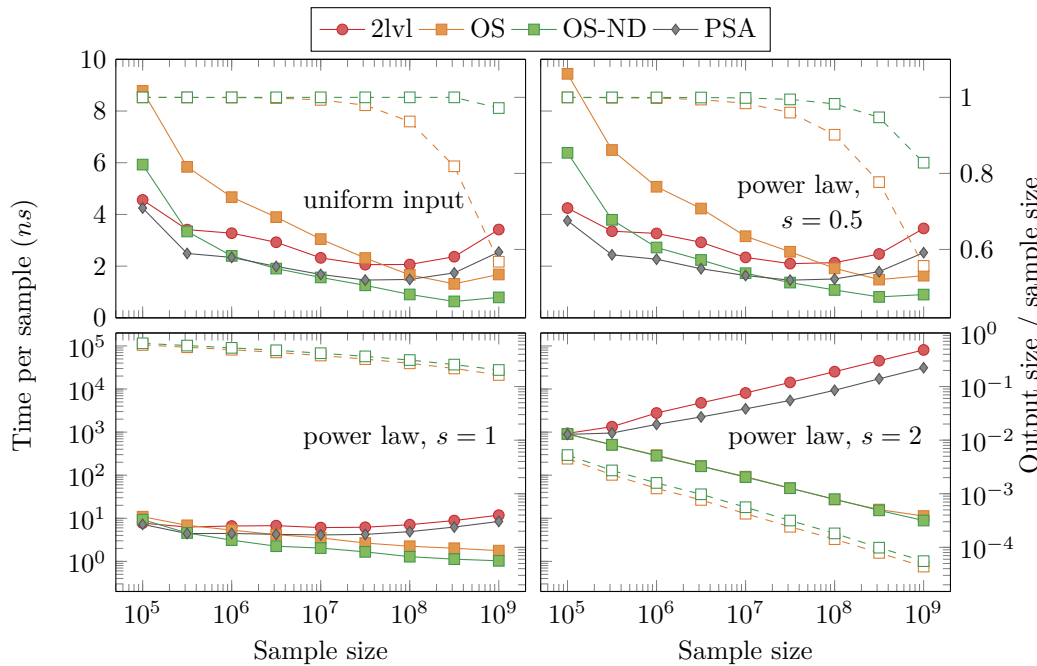
Throughput. Figure 5 shows the query throughput of the different approaches. We can see that 2lvl suffers a significant slowdown compared to PSA on all inputs since an additional query for a meta-item is needed (this is also clearly visible in Figure 4). This slowdown is much more pronounced on the Intel machine (60%) than on the AMD machine (30%) because inter-NUMA-node memory access latency on the quad-socket Intel machine is much higher than on the single-socket AMD machine. Nonetheless, throughput is limited by the latency of random accesses to memory for the (bottom) tables for both approaches and on both machines.

(a) Intel machine (158 threads). Note the logarithmic y axes for the bottom plots.(b) AMD machine (62 threads). Note the logarithmic y axes for the bottom plots.

■ **Figure 5** Query throughput of the different methods for $n = 10^9$, using all available cores. Top left: uniform inputs, top right: power law with $s = 0.5$, bottom left: power law $s = 1$, bottom right: power law $s = 2$. Dashed lines on the right y axis belong to the same-colored solid lines on the left y axis and show fraction of output size over sample size for output-sensitive algorithms.



(a) Intel machine (158 threads). Note the logarithmic y axes for the bottom plots.



(b) AMD machine (62 threads). Note the logarithmic y axes for the bottom plots.

Figure 6 Time per unique output item for the different methods for $n = 10^9$ using all available cores on the Intel (top) and AMD (bottom) machines. Top left: uniform inputs, top right: power law with $s = 0.5$, bottom left: power law $s = 1$, bottom right: power law $s = 2$. Dashed lines on the right y axis belong to the same-colored solid lines on the left y axis and show fraction of output size over sample size for output-sensitive algorithms.

As long as the sample contains few duplicates (*cf.* the dashed lines with scale on the right y axis, which belong to the solid lines of the same color and marker shape), the cost of base case deduplication in OS cancels out the gains of increased memory locality. On the AMD machine, where memory access locality is less important, this results in higher throughput for 2vl than for OS for small sample sizes when inputs are not too skewed. As expected, when there are many duplicates, the output sensitive algorithms (OS and OS-ND) scale very well. Omitting base case deduplication (OS-ND) doubles throughput for uniform inputs and does no harm for skewed inputs, making OS-ND the consistently fastest algorithm. In comparison, adding sequential deduplication to normal alias tables using a fast hash table (Google’s `dense_hash_map`) takes 5.4 times longer for uniform inputs ($n = 10^8, 10^7$ samples) compared to simply storing samples in an array without deduplication.

Lastly, we observe that 2vl and PSA throughput levels off after $10^{7.5}$ samples on the AMD machine, whereas it keeps increasing slightly on the Intel machine.

Time per Item. Figure 6 shows the time per unique item in the sample. We can see that the 2vl and PSA approaches work well as long as few items have multiplicity larger than one. In these cases, what OS gains from having higher locality of memory accesses is lost in base case deduplication – especially on the AMD machine. Because it may emit items repeatedly, OS-ND does not suffer from this and is the fastest algorithm. The same is true for the power law inputs with $s = 0.5$ and $s = 1$ (observe that as in Figure 5, the y axes for the lower two plots are logarithmic). For power law inputs with $s = 2$, the running time of OS and OS-ND is nearly constant regardless of sample size. This is because the number of unique items is very low for this input (measured in the low thousands), and thus what little time is spent on sampling is dominated by thread synchronization and scheduling overhead. This overhead is particularly problematic with the 158 threads on the Intel machine (Figure 6a), where it amounts to several milliseconds, ten times as much as on the AMD machine (Figure 6b). Further, observe that the leveling off of 2vl and PSA throughput on the AMD machine causes unfavorable time per item for large samples.

11 Conclusions and Future Work

We have presented parallel algorithms for a wide spectrum of weighted sampling problems running on a variety of machine models. The algorithms are at the same time efficient in theory and sufficiently simple to be practically useful.

Future work can address the trade-off between parallel alias tables and (distributed) 2-level alias tables (fast queries versus fast scalable construction). We can also consider support of additional machine models such as GPUs as well as MapReduce or of other big data tools like Thrill [3] or Spark [42]. For example, the solution to WRS-1 based on Theorem 2-(3) could be implemented on top of Thrill using its prefix sum primitive. It might also be possible to transfer some of the distributed data structures. For example, the variant of Theorem 2-(1) could be supported by emulating the behavior of $p = \sqrt{n}$ PEs. Storing the \sqrt{n} second-level tables as elementary objects in the big data tool ensures load balancing and fault tolerance; a replicated meta-table of size \sqrt{n} can be used to assign samples to groups.

References

- 1 Joachim H. Ahrens and Ulrich Dieter. Sequential Random Sampling. *ACM Transactions on Mathematical Software (TOMS)*, 11(2):157–169, June 1985.
- 2 Richard Arratia. On the amount of dependence in the prime factorization of a uniform random integer. *Contemporary combinatorics*, 10:29–91, 2002. Page 36.

- 3 Timo Bingmann, Michael Axtmann, Emanuel Jöbstl, Sebastian Lamm, Huyen Chau Nguyen, Alexander Noe, Sebastian Schlag, Matthias Stumpp, Tobias Sturm, and Peter Sanders. Thrill: High-Performance Algorithmic Distributed Batch Data Processing with C++. In *2016 IEEE International Conference on Big Data*, pages 172–183. IEEE, 2016.
- 4 Guy E. Blelloch. Scans as primitive parallel operations. *IEEE Transactions on Computers*, 38(11):1526–1538, November 1989.
- 5 Karl Bringmann and Kasper Green Larsen. Succinct Sampling from Discrete Distributions. In *45th ACM Symposium on Theory of Computing (STOC)*, pages 775–782. ACM, 2013.
- 6 Karl Bringmann and Konstantinos Panagiotou. Efficient sampling methods for discrete distributions. *Algorithmica*, 79(2):484–508, 2017.
- 7 M. T. Chao. A general purpose unequal probability sampling plan. *Biometrika*, 69(3):653–656, 1982.
- 8 Richard Cole. Parallel Merge Sort. *SIAM Journal on Computing*, 17(4):770–785, 1988.
- 9 Luc Devroye. *Non-Uniform Random Variate Generation*. Springer, 1986.
- 10 Pavlos S Efrimidis. Weighted random sampling over data streams. In *Algorithms, Probability, Networks, and Games: Scientific Papers and Essays Dedicated to Paul G. Spirakis on the Occasion of His 60th Birthday*, pages 183–195. Springer, 2015.
- 11 Pavlos S Efrimidis and Paul G Spirakis. Fast parallel weighted random sampling. Technical Report TR99.04.02, CTI Patras, 1999.
- 12 Pavlos S Efrimidis and Paul G Spirakis. Weighted random sampling with a reservoir. *Information Processing Letters*, 97(5):181–185, 2006.
- 13 Mark Galassi, Jim Davies, James Theiler, Brian Gough, Gerard Jungmann, Patrick Alken, Michael Booth, Fabrice Rossi, and Rhys Ulerich. *GNU scientific library: reference manual*. Network Theory, 3 edition, 2009.
- 14 Torben Hagerup. Fast Parallel Generation of Random Permutations. In *18th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 405–416. Springer, 1991.
- 15 Torben Hagerup, Kurt Mehlhorn, and J Ian Munro. Maintaining discrete probability distributions optimally. In *20th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 253–264. Springer, 1993.
- 16 Lorenz Hübschle-Schneider and Peter Sanders. Communication Efficient Algorithms for Top- k Selection Problems. In *30th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 659–668. IEEE, 2016.
- 17 Intel. *Intel Math Kernel Library 2019*. Intel, 2019. URL: <https://software.intel.com/en-us/mkl-reference-manual-for-c>.
- 18 Joseph JáJá. *An Introduction to Parallel Algorithms*. Addison Wesley, 1992.
- 19 Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing. Design and Analysis of Algorithms*. Benjamin/Cummings, 1994.
- 20 Kevin J Lang. Practical algorithms for generating a random ordering of the elements of a weighted set. *Theory of Computing Systems*, 54(4):659–688, 2014.
- 21 George Marsaglia, Wai Wan Tsang, Jingbo Wang, et al. Fast generation of discrete random variables. *Journal of Statistical Software*, 11(3):1–11, 2004.
- 22 Yossi Matias, Jeffrey Scott Vitter, and Wen-Chun Ni. Dynamic generation of discrete random variates. *Theory of Computing Systems*, 36(4):329–358, 2003.
- 23 M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACMTMCS: ACM Transactions on Modeling and Computer Simulation*, 8:3–30, 1998.
- 24 Jens Maue and Peter Sanders. Engineering Algorithms for Approximate Weighted Matching. In *6th Workshop on Experimental Algorithms (WEA)*, pages 242–255. Springer, 2007.
- 25 Kurt Mehlhorn and Peter Sanders. *Algorithms and Data Structures — The Basic Toolbox*. Springer, 2008.

- 26 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- 27 Kirill Müller. Accelerating weighted random sampling without replacement. *Arbeitsberichte Verkehrs-und Raumplanung*, 1141, 2016.
- 28 Frank Olken and Doron Rotem. Random sampling from databases: a survey. *Statistics and Computing*, 5(1):25–42, 1995.
- 29 R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019. URL: <https://www.R-project.org>.
- 30 Sanguthevar Rajasekaran and John H Reif. Optimal and sublogarithmic time randomized parallel sorting algorithms. *SIAM Journal on Computing*, 18(3):594–607, 1989.
- 31 Abhiram G Ranade. How to emulate shared memory. *Journal of Computer and System Sciences*, 42(3):307–326, 1991.
- 32 Peter Sanders. On the Competitive Analysis of Randomized Static Load Balancing. In S. Rajasekaran, editor, *First Workshop on Randomized Parallel Algorithms*, Honolulu, Hawaii, 1996. <http://algo2.iti.kit.edu/sanders/papers/rand96.pdf>.
- 33 Peter Sanders. Random Permutations on Distributed, External and Hierarchical Memory. *Information Processing Letters*, 67(6):305–310, 1998.
- 34 Peter Sanders, Sebastian Lamm, Lorenz Hübschle-Schneider, Emanuel Schrade, and Carsten Dachsbacher. Efficient Random Sampling – Parallel, Vectorized, Cache-Efficient, and Online. *ACM Transaction on Mathematical Software*, 44(3):29:1–29:14, 2018.
- 35 Peter Sanders, Sebastian Schlag, and Ingo Müller. Communication Efficient Algorithms for Fundamental Big Data Problems. In *2013 IEEE International Conference on Big Data*, pages 15–23. IEEE, 2013.
- 36 Julian Shun. Improved parallel construction of wavelet trees and rank/select structures. In *2017 Data Compression Conference (DCC)*, pages 92–101. IEEE, 2017.
- 37 Michael D. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on Software Engineering (TSE)*, 17(9):972–975, 1991.
- 38 Alastair J Walker. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):253–256, 1977.
- 39 Wikichip.org. Intel Xeon Gold 6138. https://en.wikichip.org/w/index.php?title=intel/xeon_gold/6138&oldid=71062, 2019. Accessed April 26, 2019.
- 40 Chak-Kuen Wong and Malcolm C. Easton. An efficient method for weighted sampling without replacement. *SIAM Journal on Computing*, 9(1):111–113, 1980.
- 41 Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In *24th ACM Symposium on Operating Systems Principles (SOSP)*, pages 423–438. ACM, 2013.
- 42 Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache Spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

External Memory Priority Queues with Decrease-Key and Applications to Graph Algorithms

John Iacono 

Department of Computer Science, Université Libre de Bruxelles, Belgium
<http://johniacono.com/>
ulb@johniacono.com

Riko Jacob

Computer Science Department, IT University of Copenhagen, Denmark
<http://www.itu.dk/people/rikj/>
rikj@itu.dk

Konstantinos Tsakalidis 

Department of Computer Science, University of Liverpool, United Kingdom
<https://cgi.csc.liv.ac.uk/~tsakalid/>
K.Tsakalidis@liverpool.ac.uk

Abstract

We present priority queues in the external memory model with block size B and main memory size M that support on N elements, operation UPDATE (a combination of operations INSERT and DECREASEKEY) in $O\left(\frac{1}{B} \log_{\frac{M}{B}} \frac{N}{B}\right)$ amortized I/Os and operations EXTRACTMIN and DELETE in $O\left(\lceil \frac{M^\varepsilon}{B} \log_{\frac{M}{B}} \frac{N}{B} \rceil \log_{\frac{M}{B}} \frac{N}{B}\right)$ amortized I/Os, for any real $\varepsilon \in (0, 1)$, using $O\left(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B}\right)$ blocks. Previous I/O-efficient priority queues either support these operations in $O\left(\frac{1}{B} \log_2 \frac{N}{B}\right)$ amortized I/Os [Kumar and Schwabe, SPDP '96] or support only operations INSERT, DELETE and EXTRACTMIN in optimal $O\left(\frac{1}{B} \log_{\frac{M}{B}} \frac{N}{B}\right)$ amortized I/Os, however without supporting DECREASEKEY [Fadel et al., TCS '99].

We also present buffered repository trees that support on a multi-set of N elements, operation INSERT in $O\left(\frac{1}{B} \log_{\frac{M}{B}} \frac{N}{B}\right)$ I/Os and operation EXTRACT on K extracted elements in $O\left(M^\varepsilon \log_{\frac{M}{B}} \frac{N}{B} + K/B\right)$ amortized I/Os, using $O\left(\frac{N}{B}\right)$ blocks. Previous results achieve $O\left(\frac{1}{B} \log_2 \frac{N}{B}\right)$ I/Os and $O\left(\log_2 \frac{N}{B} + \frac{K}{B}\right)$ I/Os, respectively [Buchsbaum et al., SODA '00].

Our results imply improved $O\left(\frac{E}{B} \log_{\frac{M}{B}} \frac{E}{B}\right)$ I/Os for single-source shortest paths, depth-first search and breadth-first search algorithms on massive directed dense graphs (V, E) with $E = \Omega(V^{1+\varepsilon})$, $\varepsilon > 0$ and $V = \Omega(M)$, which is equal to the I/O-optimal bound for sorting E values in external memory.

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis; Theory of computation \rightarrow Shortest paths; Theory of computation \rightarrow Data structures design and analysis; Hardware \rightarrow External storage

Keywords and phrases priority queues, external memory, graph algorithms, shortest paths, depth-first search, breadth-first search

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.60

Related Version A full version of the paper is available at <https://arxiv.org/abs/1903.03147>.

Funding *John Iacono*: Supported by Fonds de la Recherche Scientifique-FNRS under Grant no MISU F 6001 1. Partially supported by NSF grants CCF-1319648 and CCF-1533564.

Konstantinos Tsakalidis: Partially supported by NSF grants CCF-1319648 and CCF-1533564.



© John Iacono, Riko Jacob, and Konstantinos Tsakalidis;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 60; pp. 60:1–60:14

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Priority queues are fundamental data structures with numerous applications across computer science, most prominently in the design of efficient graph algorithms. They support the following operations on a set of N stored *elements* of the type $(key, priority)$, where “key” serves as an identifier and “priority” is a value from a total order:

- INSERT(element e): Insert element e to the priority queue.
- DELETE(key k): Remove all elements with key k from the priority queue.
- element $e = \text{EXTRACTMIN}()$: Remove and return the element e in the priority queue with the smallest priority.
- DECREASEKEY(element (k, p)): Given that an element with key k and priority p' is stored in the priority queue, if priority $p < p'$, replace the element’s priority p' with p .

Operation UPDATE(element (k, p)) is a combination of operations INSERT and DECREASEKEY, such that if the priority queue does not contain any element with key k , INSERT((k, p)) is executed, otherwise DECREASEKEY((k, p)) is executed.

We study the problem of designing priority queues that support all these operations in external memory. In the *external memory model* (also known as the *I/O model*) [1] the amount of input data is assumed to be much larger than the main memory size M . Thus, the data is stored in an external memory device (i.e. disk) that is divided into consecutive blocks of size B elements. Time complexity is measured in terms of *I/O operations* (or *I/Os*), namely block transfers from external to main memory and vice versa, while computation in main memory is considered to be free. Space complexity is measured in the number of blocks occupied by the input data in external memory. Algorithms and data structures in this model are considered *cache-aware*, since they are parameterized in terms of M and B . In contrast, *cache-oblivious* algorithms and data structures [11] are oblivious to both these values, which allows them to be efficient along all levels of a memory hierarchy. I/O-optimally scanning and sorting x consecutive elements in an array are commonly denoted to take $\text{Scan}(x) = O\left(\frac{x}{B}\right)$ I/Os and $\text{Sort}(x) = O\left(\frac{x}{B} \log \frac{M}{B} \frac{x}{B}\right)$ I/Os, respectively [1, 11].

Priority queues are a basic component in several fundamental graph algorithms, including:

- The *single-source shortest paths (SSSP)* algorithm on directed graphs with positively weighted edges, which computes the minimum edge-weight paths from a given source node to all other nodes in the graph.
- The *depth-first search (DFS)* and *breadth-first search (BFS)* algorithms on directed unweighted graphs, which number all nodes of the graph according to a depth-first or a breadth-first exploration traversal starting from a given source node, respectively.

Another necessary component for these algorithms are I/O-efficient *buffered repository trees (BRTs)* [6, 2, 7]. They are used by external memory graph algorithms in order to confirm that a given node has already been visited by the algorithm. This avoids expensive random-access I/Os incurred by internal memory methods. In particular, BRTs support the following operations on a stored multi-set of N (key, value) elements, where “key” serves as an identifier and “value” is from a total order:

- INSERT(element e): Insert element e to the BRT.
- element $e_i = \text{EXTRACT}(\text{key } k)$: Remove and return all K elements e_i (for $i \in [1, K]$) in the BRT with key k .

■ **Table 1** Asymptotic amortized I/O-bounds of cache-aware and cache-oblivious priority queue operations (respectively, above and below the horizontal line) on N elements and real $\varepsilon \in (0, 1)$. *Expected I/Os.

	INSERT	DELETE	EXTRACTMIN	DECREASEKEY
[10]	$\frac{1}{B} \log_{\frac{M}{B}} \frac{N}{B}$	$\frac{1}{B} \log_{\frac{M}{B}} \frac{N}{B}$	$\frac{1}{B} \log_{\frac{M}{B}} \frac{N}{B}$	–
[14]	$\frac{1}{B} \log_2 \frac{N}{B}$	$\frac{1}{B} \log_2 \frac{N}{B}$	$\frac{1}{B} \log_2 \frac{N}{B}$	$\frac{1}{B} \log_2 \frac{N}{B}$
[13]*	$\frac{1}{B} \log_{\log N} \frac{N}{B}$	$\frac{1}{B} \log_{\log N} \frac{N}{B}$	$\frac{1}{B} \log_{\log N} \frac{N}{B}$	$\frac{1}{B} \log_{\log N} \frac{N}{B}$
New	$\frac{1}{B} \log_{\frac{M}{B}} \frac{N}{B}$	$\lceil \frac{M^\varepsilon}{B} \log_{\frac{M}{B}} \frac{N}{B} \rceil \log_{\frac{M}{B}} \frac{N}{B}$	$\lceil \frac{M^\varepsilon}{B} \log_{\frac{M}{B}} \frac{N}{B} \rceil \log_{\frac{M}{B}} \frac{N}{B}$	$\frac{1}{B} \log_{\frac{M}{B}} \frac{N}{B}$
[2]	$\frac{1}{B} \log_{\frac{M}{B}} N$	$\frac{1}{B} \log_{\frac{M}{B}} N$	$\frac{1}{B} \log_{\frac{M}{B}} N$	–
[5, 7]	$\frac{1}{B} \log_2 N$	$\frac{1}{B} \log_2 N$	$\frac{1}{B} \log_2 N$	$\frac{1}{B} \log_2 N$

1.1 Previous work

Designing efficient external memory priority queues able to support operation DECREASEKEY (or at least operation UPDATE) has been a long-standing open problem [14, 10, 16, 9, 7, 13]. I/O-efficient adaptations of the standard heap data structure [10] or other sorting-based approaches [16], despite achieving optimal base- (M/B) logarithmic amortized I/O-complexity, fail to support operation DECREASEKEY. (Nevertheless, we use these priority queues as subroutines in our structure.) Adaptations of the tournament tree data structure support all operations, albeit in not so efficient base-2 logarithmic amortized I/Os [14, 7]. Indeed, in the recent work of Eenberg, Larsen and Yu [9] it is shown that for a sequence of N operations, any external-memory priority queue supporting DECREASEKEY must spend $\max\{\text{INSERT}, \text{DELETE}, \text{EXTRACTMIN}, \text{DECREASEKEY}\} = \Omega\left(\frac{1}{B} \log_{\log N} B\right)$ amortized I/Os. Randomized priority queues with matching complexity were recently presented by Jiang and Larsen [13].

The BRTs introduced by Buchsbaum et al. [6, Lemma 2.1] and their cache-oblivious counterparts [2] support INSERT in $O\left(\frac{1}{B} \log_2 N\right)$ amortized I/Os and EXTRACT on K extracted elements in $O(\log_2 N + K/B)$ amortized I/Os on a multi-set of N stored elements.

1.2 Our contributions

We present I/O-efficient priority queues that support on N stored elements, operation UPDATE in optimal $O\left(\frac{1}{B} \log_{\frac{M}{B}} \frac{N}{B}\right)$ amortized I/Os and operations EXTRACTMIN and DELETE in $O\left(\lceil \frac{M^\varepsilon}{B} \log_{\frac{M}{B}} \frac{N}{B} \rceil \log_{\frac{M}{B}} \frac{N}{B}\right)$, for any real $\varepsilon \in (0, 1)$. Our priority queues are the first to support operation UPDATE (and thus DECREASEKEY) in a cache-aware setting in optimal I/Os, while also I/O-optimally supporting operation INSERT. These bounds improve upon previous priority queues supporting DECREASEKEY [14], albeit at the expense of suboptimal I/O-efficiency for EXTRACTMIN and DELETE (respecting the lower bound of [9] for $M = \Omega(B \log_2 N)$). See Table 1 for a comparison with previous cache-aware and cache-oblivious I/O-efficient priority queues.

We also present I/O-efficient BRTs that support on a multi-set of N elements, operation INSERT in $O\left(\frac{1}{B} \log_{\frac{M}{B}} \frac{N}{B}\right)$ amortized I/Os and operation EXTRACT on K extracted elements in $O\left(M^\varepsilon \log_{\frac{M}{B}} \frac{N}{B} + K/B\right)$ amortized I/Os. Previous cache-aware bounds were $O\left(\frac{1}{B} \log_2 \frac{N}{B}\right)$ and $O\left(\log_2 \frac{N}{B} + \frac{K}{B}\right)$, respectively [6]. Combined with our priority queues, for external memory SSSP, DFS and BFS algorithms on graphs with V nodes and E directed edges, we achieve $O\left(V \frac{M^{\frac{\alpha}{1+\alpha}}}{B} \log_{\frac{M}{B}}^2 \frac{E}{B} + V \log_{\frac{M}{B}} \frac{E}{B} + \frac{E}{B} \log_{\frac{M}{B}} \frac{E}{B}\right)$ I/Os. This compares to previous

$O\left(\left(V + \frac{E}{B}\right) \log_2 E\right)$ I/Os for directed SSSP [14, 15, 7] and $O\left(\left(V + \frac{E}{B}\right) \log_2 \frac{V}{B} + \frac{E}{B} \log_{\frac{M}{B}} \frac{E}{B}\right)$ I/Os for directed DFS and BFS [6, 2]. Our bounds are I/O-optimal for dense graphs with $E = \Omega(V^{1+\epsilon})$ and $V = \Omega(M)$.

1.3 Our approach

The main component of our priority queues is the *x-treap*, a recursive structure inspired by similar cache-oblivious *x-box* [4] and cache-aware hashing data structures [12] that solve the dynamic dictionary problem in external memory (respectively, under predecessor and membership queries on a dynamic set of keys). To solve the priority queue problem, we adapt this recursive scheme to also handle priorities, inspired by the cache-oblivious priority queues of Brodal et al. [5] that support UPDATE, yet in suboptimal I/Os. Here we discuss informally these ideas, the rationale for combining them, and a back-of-the-envelope intuitive, but incomplete analysis. It is hoped that this will provide the intuition to more easily follow the full details in the sequel.

The idea behind the cache-oblivious priority queues of Brodal et al. [5] is simple. The structure has a logarithmic number of levels, where level i has two arrays, or buffers, of size roughly 2^i . These buffers are called the *front* and *rear* buffers. They contain key-priority pairs or a key-delete message (described later). The idea is that the front buffers are sorted, with everything in the i -th front buffer having smaller priorities than everything in the $(i + 1)$ -th front buffer. The items in the rear buffers do not have this rigorous ordering, but instead must be larger than the items in the rear buffer at the smaller levels. When an UPDATE operation occurs, the key-priority pair gets placed in the first rear buffer; when an EXTRACTMIN operation occurs, the key-priority pair with the smallest priority is removed from the first front buffer. Every time a level- i buffer gets too full or empty relative to its target size of 2^i , this is fixed by moving things up or down as needed, and moving things from the rear to front buffer if that respects the ordering of items in the front buffer. This resolution of problems is done efficiently using a scan of the affected and neighbouring levels. Thus, looking in a simplified manner at the lifetime of an UPDATED item, it will be inserted in the smallest rear buffer, be pushed down to larger rear buffers as they overflow, be moved from a rear buffer to a front buffer once it has gone down to a level where its priority is compatible with those in the corresponding front buffer, then moves up from the front buffer to smaller front buffers as they underflow, and is finally removed from the smallest front buffer during an EXTRACTMIN. Thus, during its lifetime, it could be moved from one level to another a total of $O\left(\log_2 \frac{N}{B}\right)$ times at an I/O-cost of $O\left(\frac{1}{B}\right)$ per level, for a total cost of $O\left(\frac{1}{B} \log_2 \frac{N}{B}\right)$ I/Os. One detail is that when an item moves from a rear to a front buffer, we want to make sure that no items in larger levels with the same key and larger priority are ever removed. This is done through special delete messages, which stay in the rear buffers and percolate down, removing any key-priority pairs with the given key that they encounter in their buffer or the corresponding front buffer.

The problem with this approach is that the base-2 logarithm seems unavoidable, with the simple idea of a geometrically increasing buffer size. So here instead we use the more complicated recursion introduced with the cache-oblivious *x-box* [4] structure and also used in the cache-aware hashing data structures [12]. In its simplest form, used for a dictionary, an *x-box* has three buffers: top, middle and bottom (respectively of approximate size x , $x^{1.5}$ and x^2), as well as \sqrt{x} recursive *upper-level* \sqrt{x} -boxes (ordered logically between the top and middle buffers) and x recursive *lower-level* \sqrt{x} -boxes (ordered logically between the middle and bottom buffers). Data in each buffer is sorted, and all keys in a given recursive buffer are

smaller than all keys in subsequent recursive buffers in the same level (upper or lower). There is no enforced order among keys in different buffers or in a recursive upper- or lower-level \sqrt{x} -box. The key feature of this construction is that the top/middle/bottom buffers have the same size as the neighbouring recursive buffers: the top buffer has size x , the top buffers of the upper-level recursive \sqrt{x} -boxes have total size x ; the middle buffer, sum of the bottom buffers of the upper-level, and sum of the top buffers of the lower-level recursive structures all have size $x^{1.5}$; the sum of the bottom buffers of the lower-level recursive structures and the bottom buffer both have size x^2 . Therefore, when for example a top buffer overflows, it can be fixed by moving excess items to the top buffers of the top recursive substructures. In a simplified view with only insertions, as buffers overflow, an item over its lifetime will percolate from the top buffer to the upper-level substructures, to the middle buffer, to the lower-level substructures, and to the bottom buffer, with each overflow handled only using scans. Assuming a base case of size M , there will be $O(\log_M N)$ times that an item will move from one buffer to another and an equal number of times that an item will pass through a base case. One major advantage of this recursive approach, is that an item will pass through a small base case not just once at the structure's top, as before, but many times.

We combine these ideas to form the x -treap, described at a high level as follows: Everywhere an x -box has a buffer, we replace it with front and rear buffers storing key-priority pairs. The order used by x -box is imposed on the keys, not the priorities. The order imposed on priorities in the Brodal et al. structure are carried over and imposed on the priorities in different levels of the x -treap; this is aided by the fact that the buffers in the x -treap form a DAG, thus the buffers where items with a given key can appear, form a natural total order. Hence, this forms a treap-like arrangement where we use the keys for order in one dimension and priorities for order in the other. We use a separate trivial base case structure which is invoked at a size smaller than the memory size; it stores items in no particular order and thus supports fast insertion of items when a neighbouring buffer adds them ($O(\frac{1}{B})$), but slow ($O(M^\epsilon)$ amortized) removal of items with small priorities to fix the underflow of a front buffer above. Thus, again considering the typical hypothetical lifetime of an item, it will be inserted at the top in the rear buffer, percolate down $O(\log_{\frac{M}{B}} \frac{N}{B})$ levels and base cases at a cost of $O(\frac{1}{B})$ amortized each, move over to a front buffer, then percolate up $O(\log_{\frac{M}{B}} \frac{N}{B})$ levels at a cost of $O(\frac{M^\epsilon}{B})$ amortized each. Thus, the total amortized cost for an item that is eventually removed by an EXTRACTMIN is $O(\frac{M^\epsilon}{B} \log_{\frac{M}{B}} \frac{N}{B})$.

However, we want the amortized cost for an item that is inserted via UPDATE to be much faster than this, i.e. $O(\frac{1}{B} \log_{\frac{M}{B}} \frac{N}{B})$. This requires additional observations and tricks. The first is that, unlike Brodal et al., we do not use delete-type messages that percolate down to eliminate items with larger than minimum priority in order to prevent their removal from EXTRACTMIN. Instead, we adopt a much simpler approach, and use a hash table to keep track of all keys that have been removed by an EXTRACTMIN, and when an EXTRACTMIN returns a key that has been seen before, it is discarded and EXTRACTMIN is repeated. The second trick is to simply ensure that each buffer has at most one item with each key (and removes key-priority pairs other than the one with the minimum priority among those with the same key in the buffer). This has the effect that if there are a total of u updates performed on a key before it is removed by an EXTRACTMIN, the total cost will involve up to $O(u \log_{\frac{M}{B}} \frac{N}{B})$ percolations down at a cost of $O(\frac{1}{B})$, but only $O(\log_{\frac{M}{B}}^2 \frac{N}{B})$ percolations up at a cost of $O(\frac{M^\epsilon}{B})$ amortized each. After the EXTRACTMIN, some items may still remain in the structure and will be discarded when removed by EXTRACTMIN however, due to the no-duplicates-per-level property there will only be $O(\log_{\frac{M}{B}} N)$ such items (called *ghosts*) which

will incur a cost of at most $O\left(\lceil \frac{M^\epsilon}{B} \log_{\frac{M}{B}} \frac{N}{B} \rceil\right)$ amortized each, where the ceiling accounts for accessing the hash table. Thus the total amortized cost for the lifetime of the u UPDATES and one EXTRACTMIN involving a single key is $O\left(\frac{u}{B} \log_{\frac{M}{B}} \frac{N}{B} + \lceil \frac{M^\epsilon}{B} \log_{\frac{M}{B}} \frac{N}{B} \rceil \log_{\frac{M}{B}} \frac{N}{B}\right)$. This cost can be apportioned in the amortized sense by having the EXTRACTMIN cost $O\left(\lceil \frac{M^\epsilon}{B} \log_{\frac{M}{B}} \frac{N}{B} \rceil \log_{\frac{M}{B}} \frac{N}{B}\right)$ amortized and the updates cost $O\left(\frac{1}{B} \log_{\frac{M}{B}} \frac{N}{B}\right)$ amortized each, assuming that the treap finishes in an empty state and no item can be UPDATED after it has been EXTRACTMIN'd.

The details that implement these rough ideas consume the rest of the paper. One complication that eludes the above discussion is that items don't just percolate down and then up; they could move up and down repeatedly and this can be handled through an appropriate potential function. The various layers of complexity needed for the x -treap recursion combined with the front/rear buffer idea, various types of over/underflows of buffers, a special base case, having the middle and bottom buffers be of size $x^{1+\frac{\alpha}{2}}$ and $x^{1+\alpha}$ for a suitable parameter α rather than $x^{1.5}$ and x^2 as described above, and a duplicate-catching hash table, result in a complex structure with an involved potential analysis, but that follows naturally from the above high-level description.

2 x -Treap

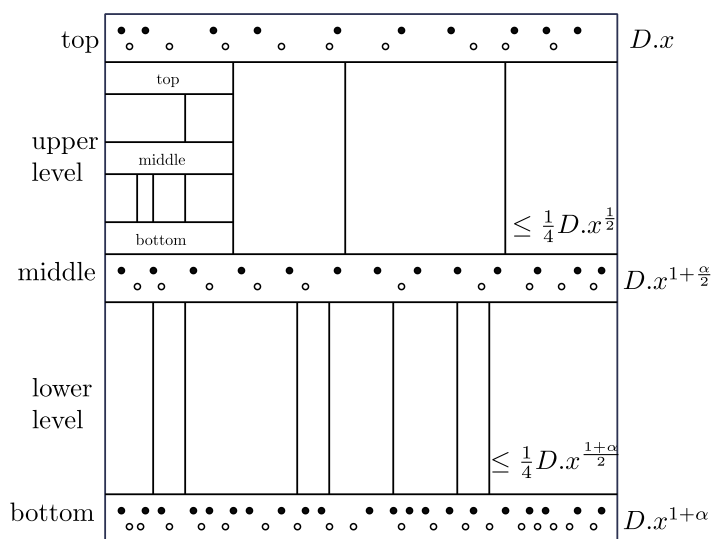
Given real parameter $\alpha \in (0, 1]$ and *key range* $[k_{\min}, k_{\max}) \subseteq \mathbb{R}$, an x -treap D stores a set of at most $2(D.x)^{1+\alpha}$ *elements* $(*, k, p)$ associated with a *key* $k \in [D.k_{\min}, D.k_{\max})$ and a *priority* p from a totally ordered set. D represents a set $D.rep$ of pairs (key, priority), such that a particular key k contained in D is represented to have the smallest priority p of any element with key k stored in D , unless an element with key k and a smaller priority has been removed from the structure. In particular, we call the key and priority *represented*, when the pair (key, priority) $\in D.rep$. A *representative* element contains a represented key and its represented priority. More formally, we define:

$$D.rep := \bigcup_{\{k|(k,p) \in D\}} \left\{ \left(k, \min_p (k, p) \in D \right) \right\}$$

The proposed representation scheme works under the assumption that a key that is not represented by the structure anymore, cannot become represented again. In other words, a key returned by operation EXTRACTMIN cannot be INSERTED to the structure again.

The following *interface operations* are supported (See full version for their correctness):

- BATCHED-INSERT(D, e_1, e_2, \dots, e_b): For constant $c \in (0, \frac{1}{3}]$, insert $b \leq c \cdot D.x$ elements e_1, e_2, \dots, e_b to D , given they are key-sorted with keys $e_i.k \in [D.k_{\min}, D.k_{\max})$, $i \in [1, b]$. BATCHED-INSERT adds the pairs $(e_i.k, e_i.p)$ to $D.rep$ with key $e_i.k$ that is not contained in D already. BATCHED-INSERT decreases the priority of a represented key $e_i.k$ to $e_i.p$, if its represented priority is larger than $e_i.p$ before the operation. More formally, let X_{new} contain the inserted pairs $(e_i.k, e_i.p)$ with $e_i.k \notin D.rep$. Let X_{old} contain the pairs in $D.rep$ with an inserted key, but with larger priority than the inserted one, and let X_{dec} contain these inserted pairs. After BATCHED-INSERT, a new x -treap D' is created where $D'.rep = D.rep \cup X_{new} \cup X_{dec} \setminus X_{old}$.
- BATCHED-EXTRACTMIN(D): For constant $c \in (0, \frac{1}{4}]$, remove and return the at most $c \cdot D.x$ elements (k, p) with the smallest priorities in D . BATCHED-EXTRACTMIN removes the pairs X_{\min} from $D.rep$ with the at most $c \cdot D.x$ smallest priorities. Let X_{key} contain the pairs in D with keys in X_{\min} . After BATCHED-EXTRACTMIN, a new x -treap D' is created where $D'.rep = D.rep \setminus X_{\min} \setminus X_{key}$.



■ **Figure 1** Overview of an x -treap D on “key” \times “partial order” space. Black/white dots represent elements in the front/rear buffers, respectively. All buffers are resolved. Buffer sizes and maximum number of subtrees in a level are shown on the right-hand side.

► **Theorem 1.** *An x -treap D supports BATCHED-EXTRACTMIN in $O\left(M^{\frac{\alpha}{1+\alpha}} \frac{1+\alpha}{B} \log_M D.x\right)$ amortized I/Os per element and BATCHED-INSERT in $O\left(\frac{1+\alpha}{B} \log_M D.x\right)$ amortized I/Os per element, using $O\left(\frac{(D.x)^{1+\alpha}}{B} \log_M D.x\right)$ blocks, for any real $\alpha \in (0, 1]$.*

The structure is recursive. The base case is described separately in Subsection 2.3. The base case structure is used when $D.x \leq c' M^{\frac{1}{1+\alpha}}$ (for an appropriately chosen constant $c' > 0$). Thus assuming $D.x > c' M^{\frac{1}{1+\alpha}}$, we define an x -treap to contain three *buffers* (which are arrays that store elements) and many \sqrt{x} -treaps (called *subtreaps*). Specifically, the *top*, *middle* and *bottom* buffers have *sizes* $D.x$, $(D.x)^{1+\frac{\alpha}{2}}$ and $(D.x)^{1+\alpha}$, respectively. Each buffer is divided in the middle into a *front* and a *rear buffer*. The subtrees are divided into the *upper* and the *lower level* that contain at most $\frac{1}{4} (D.x)^{\frac{1}{2}}$ and $\frac{1}{4} (D.x)^{\frac{1+\alpha}{2}}$ subtrees, respectively. Let $|b|$ denote the *size* of a buffer b . We define the *capacity* of an x -treap D to be the maximum number of elements it can contain, which is $D.x + \frac{5}{4} (D.x)^{1+\frac{\alpha}{2}} + \frac{5}{4} (D.x)^{1+\alpha} < 2 (D.x)^{1+\alpha}$.

We define a partial order (\preceq) using the terminology “above/below” among the buffers of an x -treap and all of the buffers in recursive subtrees or base case structures. In this order we have top buffer \preceq upper level recursive subtrees \preceq middle buffer \preceq lower level recursive subtrees \preceq bottom buffer.

Along with all buffers of D , we also store several additional pieces of bookkeeping information: a counter with the total number of elements stored in D and an index indicating which subtree is stored in which space in memory.

2.1 Invariants

An x -treap D maintains the following invariants with respect to every one of its top/middle/bottom buffers b . The invariants hold after the execution of each interface operation, but may be violated during the execution. They allow changes to D that do not change $D.rep$.

1. The front and rear buffers of b store elements sorted by key and left-justified.
2. The front buffer’s elements’ priorities are smaller than the rear buffer’s elements’ priorities.

3. The front buffer's elements' priorities are smaller than all elements' priorities in buffers below b in D .
4. For a top or middle buffer b with key range $[b.k_{\min}, b.k_{\max})$, the r upper or lower subtrees $D_i, i \in \{1, r\}$, respectively, have distinct key ranges $[D_i.k_{\min}, D_i.k_{\max})$, such that $b.k_{\min} = D_1.k_{\min} < D_1.k_{\max} = D_2.k_{\min} < \dots < D_r.k_{\max} = b.k_{\max}$.
5. If the middle or bottom buffer b is not empty, then at least one upper or lower subtree is not empty, respectively.

2.2 Auxiliary operations

The operations BATCHED-INSERT and BATCHED-EXTRACTMIN make use of the following *auxiliary operations* (See full version for their implementation and correctness):

- Operation RESOLVE(D, b). We say that a buffer b is *resolved*, when the front and rear buffers contain elements with pairs (key,priority) (k, p) , such that k is a represented key, and when the front buffer contains those elements with smallest priorities in the buffer. To resolve b , operation RESOLVE assigns to the elements with represented keys, the key's minimum priority stored in b . Also, it removes any elements with non-represented keys from b . RESOLVE restores Invariant 2 in b , when it is temporarily violated by other (interface or auxiliary) operations that call it.
- Operation INITIALIZE(D, e_1, e_2, \dots, e_b) distributes to a new x -treap D , $\frac{1}{4}(D.x) \leq b \leq \frac{1}{2}(D.x)^{1+\alpha}$ elements $e_i, i \in [1, b]$ from a temporary array (divided in the middle into a front and a rear array, respecting Invariants 1 and 2).
- Operation FLUSH-UP(D) ensures that the front top buffer of D contains at least $\frac{1}{4}D.x$ elements (unless all buffers of D contains less elements altogether, in which case FLUSH-UP moves them all to the top front buffer of D). By Invariants 2 and 3, these are the elements in D with smallest priority.
- Operation FLUSH-DOWN(D) is called by BATCHED-INSERT on an x -treap D whose bottom buffer contains between $\frac{1}{2}(D.x)^{1+\alpha}$ and $(D.x)^{1+\alpha}$ elements. It moves to a new temporary array, at least $\frac{1}{6}(D.x)^{1+\alpha}$ and at most $\frac{2}{3}(D.x)^{1+\alpha}$ elements from the bottom buffer of D . It ensures that the largest priority elements are removed from D .
- Operation SPLIT(D) is called by BATCHED-INSERT on an x -treap D that contains between $\frac{1}{2}(D.x)^{1+\alpha}$ and $(D.x)^{1+\alpha}$ elements. It moves to a new temporary (front and rear) array, the at most $\frac{1}{3}(D.x)^{1+\alpha}$ elements with largest keys in D .

2.3 Base case

The x -treap is a recursive structure. When the x -treap stores few enough elements so that it can be stored in internal memory, we use simple arrays to support the interface operations and operation FLUSH-UP.

► **Lemma 2.** *An $O\left(M^{\frac{1}{1+\alpha}}\right)$ -treap fits in internal memory and supports operation BATCHED-INSERT in $O(1/B)$ amortized I/Os per element and operations BATCHED-EXTRACTMIN and FLUSH-UP in $\text{Scan}\left(M^{\frac{\alpha}{1+\alpha}}\right)$ amortized I/Os per element.*

Proof. For a universal positive constant c_0 and a constant parameter $c' < c_0^{\frac{1}{\alpha}+1}$, we allocate an array of size $\left(c' \left(M^{\frac{1}{1+\alpha}}\right)\right)^{\frac{\alpha}{1+\alpha}} \leq c_0 M$ and divide it in the middle into a front and a rear buffer that store elements and maintain only Invariants 1 and 2. To implement BATCHED-INSERT on at most $\frac{c'}{2}M^{\frac{1}{1+\alpha}}$ elements, we simply add them to the rear buffer and

update the counter. This costs $O\left(\frac{M^{\frac{1}{1+\alpha}}}{B}/\frac{1}{2}M^{\frac{1}{1+\alpha}}\right) = O\left(\frac{1}{B}\right)$ I/Os amortized per added element, since we only scan the part of the rear buffer where the elements are being added to. To implement BATCHED-EXTRACTMIN on at most $\frac{c'}{2}M^{\frac{1}{1+\alpha}}$ extracted elements, we RESOLVE the array (as implemented for Theorem 1), remove and return all elements in the front buffer, and update the counter. By Lemma 5 (proven later in Subsection 2.5) this costs $O\left(\frac{M}{B}/\frac{1}{2}M^{\frac{1}{1+\alpha}}\right) = O\left(\frac{M^{\frac{\alpha}{1+\alpha}}}{B}\right)$ I/Os amortized per extracted element. FLUSH-UP is implemented like BATCHED-EXTRACTMIN with the difference that the returned elements are not removed from the array. ◀

2.4 Interface operations

(See full version for the correctness of the interface operations.)

2.4.1 Inserting elements to an x -treap

Interface operation BATCHED-INSERT on an x -treap D is implemented by means of the recursive subroutine BATCHED-INSERT($D, e_1, \dots, e_{c \cdot |b|}, b$) that also takes as argument a top or middle buffer b of D and inserts $c \cdot |b|$ elements $e_1, \dots, e_{c \cdot |b|}$ (contained in a temporary array) inside and below b in D , for constant $c \in (0, \frac{1}{3}]$. For a bottom buffer b , a non-recursive subroutine BATCHED-INSERT(D, b) simply executes Step 1 below and discards the temporary array. BATCHED-INSERT($D, e_1, \dots, e_{c \cdot |b|}, b$) is implemented as following:

1. If $D.x > c' M^{\frac{1}{1+\alpha}} + c|b|$:
 - 1.1. 2-way merge into the temporary array, the elements in the temporary array and in the rear buffer of b (by simultaneous scans in increasing key-order). RESOLVE b considering the temporary array as the rear buffer of b .
 - 1.2. Implicitly partition the front buffer of b and the temporary array by the key ranges of the subtrees immediately below b . Consider the subtrees in increasing key-order by reading the index of D . For every key range (associated with subtree D') that contains at least $\frac{1}{3}(D.x)^{\frac{1}{2}}$ elements in either the front buffer of b or the temporary array: While the key range in the front buffer of b and in the temporary array contains at most $\frac{2}{3}(D.x)^{\frac{1}{2}}$ elements, do:
 - 1.2.1. Find the $\left(\frac{2}{3}(D.x)^{\frac{1}{2}}\right)$ -th smallest priority within the key range in the front buffer of b and in the temporary array (by an external memory order-statistics algorithm [3]) and move the elements in the key range with larger priority to a new auxiliary array (by simultaneous scans in increasing key-order).
 - 1.2.2. If the counter of D' plus the auxiliary array's size does not exceed the capacity of D' : BATCHED-INSERT the elements in the auxiliary array to the top buffer of D' . Discard the auxiliary array.
 - 1.2.3. Else, if there are fewer than the maximum allowed number of subtrees in the level immediately below b : SPLIT D' . Let k be the smallest key in the array returned by SPLIT (determined by a constant number of random accesses to the leftmost elements in the returned front/rear array). Move the elements in the auxiliary array with key smaller than k to a new temporary array (by a scan), BATCHED-INSERT these elements to D' and discard this temporary array. 2-way merge the remaining elements in the auxiliary array into the returned rear array and discard the auxiliary array. INITIALIZE a new subtree with the elements in the returned array. Discard the returned array.

1.2.4. Else, FLUSH-DOWN all subtrees immediately below b , which writes them to many returned arrays. 2-way merge into a new temporary array, all elements in b and in all returned arrays (by simultaneous scans in increasing key-order). (When the scan on a subtree's temporary array is over, determine the subtree with the key-next elements in the level by reading the index of D .) BATCHED-INSERT the elements in the new temporary array to the buffer b' immediately below b . Discard the new temporary array and all returned arrays.

1.3. Discard the temporary array and update the counter of D .

1.4. Else if $D.x \leq c'M^{\frac{1}{1+\alpha}} + c|b|$: BATCHED-INSERT the elements to the base case structure.

2.4.2 Extracting minimum-priority elements from an x -treap

Interface operation BATCHED-EXTRACTMIN on an x -treap D is implemented as following:

1. If $D.x > c'M^{\frac{1}{1+\alpha}}$:
 - 1.1 If the front top buffer contains less than $\frac{1}{4}D.x$ elements: FLUSH-UP the top buffer.
 - 1.2 Remove and return all the elements $(e_i.k, e_i.p)$ from the front top buffer.
 - 1.3 Update the counter of D .
2. Else if $D.x \leq c'M^{\frac{1}{1+\alpha}}$: BATCHED-EXTRACTMIN the base case structure.

2.5 Analysis

(See full version for the proofs of Lemmata 3, 4, 5, 6 and 7, respectively.)

► **Lemma 3.** *An x -treap D has $O\left(\log_{\frac{M}{B}} D.x\right)$ levels and occupies $O\left((D.x)^{1+\alpha} \log_{\frac{M}{B}} D.x\right)$ blocks of space.*

► **Lemma 4.** *By the tall-cache assumption, scanning the buffers of an x -treap D and randomly accessing $O\left((D.x)^{\frac{1+\alpha}{2}}\right)$ subtrees takes $\text{Scan}\left((D.x)^{1+\alpha}\right)$ I/Os, for any real $\alpha \in (0, 1]$.*

A buffer b_i at level $i \leq h = O\left(\log_{\frac{M}{B}} D.x\right)$ with current number of elements in the front and rear buffers b_f, b_r , respectively, has potential $\Phi(b_i) = \Phi_f(b_i) + \Phi_r(b_i)$, such that (for constants $\varepsilon := \frac{\alpha}{1+\alpha}$ and $c_0 \geq 1$):

$$\begin{aligned} \blacksquare \quad \Phi_f(b_i) &= \begin{cases} 0, & \text{if } \frac{1}{4}|b_i| \leq b_f \leq \frac{1}{3}|b_i|, \\ \frac{c_0}{B} M^\varepsilon \cdot \left(\frac{|b_i|}{4} - b_f\right) \cdot (h - i), & \text{if } b_f < \frac{1}{4}|b_i|, \\ \frac{c_0}{B} \cdot \left(b_f - \frac{|b_i|}{3}\right) \cdot (h - i), & \text{if } b_f > \frac{1}{3}|b_i|, \end{cases} \\ \blacksquare \quad \Phi_r(b_i) &= \begin{cases} 2\frac{c_0}{B} \cdot \left(b_r - \frac{|b_i|}{2}\right) \cdot (h - i), & \text{if } b_r > 0. \end{cases} \end{aligned}$$

In general, a particular element will be added to a rear buffer and will be moved down the levels of the structure over rear buffers by operation FLUSH-DOWN. A RESOLVE operation will move the element from the rear to the front buffer, if it is a representative element. From this point, it will be moved up the levels over front buffers by operation FLUSH-UP. If it is not representative, it will either get discarded by RESOLVE (when there is an element with the same key and with smaller priority in the same buffer) or it will keep going down the structure. Since RESOLVE leaves only one element per key at the level it operates, $O\left(\log_{\frac{M}{B}} D.x\right)$ elements with the same key (i.e. at most one per level) will remain in the structure after the extraction of the representative element for this key.

The M^ε -factor accounts for the extra cost of FLUSH-UP and BATCHED-EXTRACTMIN, the $(h - i)$ -factor allows for moving elements up or down a level by FLUSH-UP and FLUSH-DOWN and the 2-factor accounts for moving elements from the rear to the front buffer.

► **Lemma 5.** RESOLVE on a buffer b_i takes $\text{Scan}(|b_i|) + O(1)$ amortized I/Os.

► **Lemma 6.** BATCHED-INSERT on an x -treap D takes $O\left(\frac{1+\alpha}{B} \log_{\frac{M}{B}} D.x\right)$ amortized I/Os per element, for any real $\alpha \in (0, 1]$.

► **Lemma 7.** BATCHED-EXTRACTMIN on an x -treap D takes $O\left(M^{\frac{\alpha}{1+\alpha}} \frac{1+\alpha}{B} \log_{\frac{M}{B}} D.x\right)$ amortized I/Os per element, for any real $\alpha \in (0, 1]$.

3 Priority queues

Priority queues support operations UPDATE and EXTRACTMIN that are defined similarly to BATCHED-INSERT and BATCHED-EXTRACTMIN, respectively, but on a *single* element.

To support these operations, we compose a priority queue out of its batched counterpart in Theorem 1. The data structure on N elements consists of $1 + \log_{1+\alpha} \log_2 N$ x -treaps of doubly increasing size with parameter α being set the same in all of them. Specifically, for $i \in \{0, \log_{1+\alpha} \log_2 N\}$, the i -th x -treap D_i has $D_i.x = 2^{(1+\alpha)^i}$. We store all keys returned by EXTRACTMIN in a hash table X [12, 8].

For $i \in \{0, \log_{1+\alpha} \log_2 N - 1\}$, we define the top buffer of D_i to be “below” the bottom buffer of D_{i-1} and the bottom buffer of D_i to be “above” the top buffer of D_{i+1} . We define the set of represented pairs (key, priority) $rep = \bigcup_{i=0}^{\log_{1+\alpha} \log_2 N} D_i.rep \setminus \{(k, p) \mid k \in X\}$ and call *represented* the keys and priorities in rep . We maintain the invariant that the maximum represented priority in $D_i.rep$ is smaller than the smallest represented priority below it.

To implement UPDATE on a pair (key, priority) $\in rep$, we BATCHED-INSERT the corresponding element to D_0 . D_0 handles single-element batches, since for $i = 0 \Rightarrow x = \Theta(1)$. When D_i reaches capacity (i.e. contains $(D_i.x)^{1+\alpha}$ elements), we call FLUSH-DOWN on it, BATCHED-INSERT the elements in the returned temporary array to D_{i+1} and discard the array. This process terminates at the first x -treap that can accommodate these elements without reaching capacity.

To implement EXTRACTMIN, we call BATCHED-EXTRACTMIN to the first x -treap D_i with a positive counter, add the extracted elements to the (empty) bottom front buffer of D_{i-1} and repeat this process on D_{i-1} , until D_0 returns at least one element. If the returned key does not belong to X , we insert it. Else, we discard the element and repeat EXTRACTMIN.

To implement DELETE of a key, we add the key to X .

(See full version for the proof of Theorem 8.)

► **Theorem 8.** There exist priority queues on N elements that support operation UPDATE in $O\left(\frac{1}{B} \log_{\frac{M}{B}} \frac{N}{B}\right)$ amortized I/Os per element, operations EXTRACTMIN and DELETE in amortized $O\left(\lceil \frac{M^{1+\alpha}}{B} \log_{\frac{M}{B}} \frac{N}{B} \rceil \log_{\frac{M}{B}} \frac{N}{B}\right)$ I/Os per element, using $O\left(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B}\right)$ blocks, for any real $\alpha \in (0, 1]$.

4 Buffered repository trees

A *buffered repository tree (BRT)* [6, 2, 7] stores a multi-set of at most N elements, each associated with a *key* in the range $[1 \dots k_{\max}]$. It supports the operations INSERT and EXTRACT that, respectively, insert a new element to the structure and remove and report all

elements in the structure with a given key. To implement a BRT, we make use of the x -box [4]. Given positive real $\alpha \leq 1$ and key range $[k_{\min}, k_{\max}] \subseteq \mathfrak{R}$, an x -box D stores a set of at most $\frac{1}{2}(D.x)^{1+\alpha}$ elements associated with a key $k \in [D.k_{\min}, D.k_{\max}]$. An x -box supports the following operations:

- **BATCHED-INSERT**(D, e_1, e_2, \dots, e_b): For constant $c \in (0, \frac{1}{2}]$, insert $b \leq c \cdot D.x$ elements e_1, e_2, \dots, e_b to D , given they are key-sorted with keys $e_i.k \in [D.k_{\min}, D.k_{\max}]$, $i \in [1, b]$.
- **SEARCH**(D, κ): Return pointers to all elements in D with key κ , given they exist in D and $\kappa \in [D.k_{\min}, D.k_{\max}]$.

To implement operation **EXTRACT**(D, κ) that extracts all elements with key κ from an x -box D , we **SEARCH**(D, κ) and remove from D all returned pointed elements.

The BRT on N elements consists of $1 + \log_{1+\alpha} \log_2 N$ x -boxes of doubly increasing size with parameter α being set the same in all of them. We obtain the stated bounds by modifying the proof of the x -box [4, Theorem 5.1] to account for Lemmata 9 and 10.

► **Lemma 9.** For $D.x = \Omega\left(M^{\frac{1}{1+\alpha}}\right)$, an x -box supports operation **BATCHED-INSERT** in amortized $O\left(\frac{1+\alpha}{B} \log_{\frac{M}{B}} \frac{D.x}{B}\right)$ I/Os and operation **EXTRACT** on K extracted elements in amortized $O\left((1+\alpha) \log_{\frac{M}{B}} \frac{D.x}{B} + \frac{K}{B}\right)$ I/Os, using $O\left(\frac{(D.x)^{1+\alpha}}{B}\right)$ blocks of space.

Proof. Regarding **BATCHED-INSERT** on a cache-aware x -box, we obtain $O\left(\frac{1+\alpha}{B} \log_{\frac{M}{B}} \frac{D.x}{B}\right)$ amortized I/Os by modifying the proof of **BATCHED-INSERT** [4, Theorem 4.1] according the proof of Lemma 6. Specifically, every element is charged $O(1/B)$ amortized I/Os, instead of $O\left(1/B^{\frac{1}{1+\alpha}}\right)$, and the recursion stops when $D.x = O\left(M^{\frac{1}{1+\alpha}}\right)$, instead of $D.x = O\left(B^{\frac{1}{1+\alpha}}\right)$.

Regarding **SEARCHING** for the first occurrence of a key in a cache-aware x -box, we obtain $O\left(\log_{\frac{M}{B}} \frac{D.x}{B}\right)$ amortized I/Os by modifying the proof of **SEARCH** [4, Lemma 4.1], such that the recursion stops when $D.x = O\left(M^{\frac{1}{1+\alpha}}\right)$, instead of $D.x = O\left(B^{\frac{1}{1+\alpha}}\right)$. To **EXTRACT** all K occurrences of the searched key, we access them by scanning the x -box and by following fractional cascading pointers, which incurs an extra $O(K/B)$ I/Os. ◀

► **Lemma 10.** An $O\left(M^{\frac{1}{1+\alpha}}\right)$ -box fits in internal memory and supports operations **BATCHED-INSERT** in $O(1/B)$ amortized I/Os per element and operation **EXTRACT** on K extracted elements in $\text{Scan}\left(M^{\frac{\alpha}{1+\alpha}}\right)$ amortized I/Os per element.

Proof. We allocate an array of size $O(M)$ and implement **BATCHED-INSERT** by simply appending the inserted element to the array and **EXTRACT** by scanning the array and removing and returning all occurrences of the searched key. ◀

► **Theorem 11.** There exist buffered priority trees on a multi-set of N elements and of K extracted elements that support operations **INSERT** and **EXTRACT** in amortized $O\left(\frac{1}{B} \log_{\frac{M}{B}} \frac{N}{B}\right)$ and $O\left(\frac{M^{\frac{\alpha}{1+\alpha}}}{B} \log_{\frac{M}{B}} \frac{N}{B} + \frac{K}{B}\right)$ I/Os per element, using $O\left(\frac{N}{B}\right)$ blocks, for any real $\alpha \in (0, 1]$.

5 Graph algorithms

► **Theorem 12.** Single source shortest paths on a directed graph with V nodes and E edges can be computed in $O\left(V \frac{M^{\frac{1+\alpha}{1+\alpha}}}{B} \log_{\frac{M}{B}}^2 \frac{E}{B} + V \log_{\frac{M}{B}} \frac{E}{B} + \frac{E}{B} \log_{\frac{M}{B}} \frac{E}{B}\right)$ I/Os, for any real $\alpha \in (0, 1]$.

Proof. The algorithm of Vitter [15] (described in detail in [7, Lemma 4.1] for the cache-oblivious model) makes use of a priority queue that supports the UPDATE operation and of a BRT on $O(E)$ elements. Specifically, it makes V calls to EXTRACTMIN and E calls to UPDATE on the priority queue and V calls to EXTRACT and E calls to INSERT on the BRT. Hence, we obtain the stated bounds, by using Theorems 8 and 11. ◀

▶ **Theorem 13.** *Depth-first search and breadth-first search numbers can be assigned to a directed graph with V nodes and E edges in $O\left(V \frac{M^{1+\alpha}}{B} \log^2 \frac{E}{M} + V \log \frac{E}{M} + \frac{E}{B} \log \frac{E}{M}\right)$ I/Os, for any real $\alpha \in (0, 1]$.*

Proof. The algorithm of Buchsbaum et al. [6] makes use of a priority queue and of a BRT on $O(E)$ elements. Specifically, it makes $2V$ calls to EXTRACTMIN and E calls to INSERT on the priority queue and $2V$ calls to EXTRACT and E calls to INSERT on the BRT [6, Theorem 3.1]. Hence, we obtain the stated bounds, by using Theorems 8 and 11. ◀

References

- 1 Alok Aggarwal and S. Vitter, Jeffrey. The Input/Output Complexity of Sorting and Related Problems. *Commun. ACM*, 31(9):1116–1127, September 1988. doi:10.1145/48529.48535.
- 2 Lars Arge, Michael A. Bender, Erik D. Demaine, Bryan Holland-Minkley, and J. Ian Munro. An Optimal Cache-Oblivious Priority Queue and Its Application to Graph Algorithms. *SIAM Journal on Computing*, 36(6):1672–1695, 2007. doi:10.1137/S0097539703428324.
- 3 Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time Bounds for Selection. *J. Comput. Syst. Sci.*, 7(4):448–461, August 1973. doi:10.1016/S0022-0000(73)80033-9.
- 4 Gerth Stølting Brodal, Erik D. Demaine, Jeremy T. Fineman, John Iacono, Stefan Langerman, and J. Ian Munro. Cache-oblivious Dynamic Dictionaries with Update/Query Tradeoffs. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '10*, pages 1448–1456, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=1873601.1873718>.
- 5 Gerth Stølting Brodal, Rolf Fagerberg, Ulrich Meyer, and Norbert Zeh. Cache-Oblivious Data Structures and Algorithms for Undirected Breadth-First Search and Shortest Paths. In Torben Hagerup and Jyrki Katajainen, editors, *Algorithm Theory - SWAT 2004*, pages 480–492, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
- 6 Adam L. Buchsbaum, Michael Goldwasser, Suresh Venkatasubramanian, and Jeffery R. Westbrook. On External Memory Graph Traversal. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '00*, pages 859–860, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=338219.338650>.
- 7 Rezaul A. Chowdhury and Vijaya Ramachandran. Cache-Oblivious Buffer Heap and Cache-Efficient Computation of Shortest Paths in Graphs. *ACM Trans. Algorithms*, 14(1):1:1–1:33, January 2018. doi:10.1145/3147172.
- 8 Alex Conway, Martín Farach-Colton, and Philip Shilane. Optimal Hashing in External Memory. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 39:1–39:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2018.39.
- 9 Kasper Eenberg, Kasper Green Larsen, and Huacheng Yu. DecreaseKeys Are Expensive for External Memory Priority Queues. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 1081–1093, New York, NY, USA, 2017. ACM. doi:10.1145/3055399.3055437.

- 10 R. Fadel, K. V. Jakobsen, J. Katajainen, and J. Teuhola. Heaps and Heapsort on Secondary Storage. *Theor. Comput. Sci.*, 220(2):345–362, June 1999. doi:10.1016/S0304-3975(99)00006-7.
- 11 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-Oblivious Algorithms. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science, FOCS '99*, pages 285–, Washington, DC, USA, 1999. IEEE Computer Society. URL: <http://dl.acm.org/citation.cfm?id=795665.796479>.
- 12 John Iacono and Mihai Pătraşcu. Using Hashing to Solve the Dictionary Problem. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '12*, pages 570–582, Philadelphia, PA, USA, 2012. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=2095116.2095164>.
- 13 Shunhua Jiang and Kasper Green Larsen. A Faster External Memory Priority Queue with DecreaseKeys. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1331–1343. SIAM, 2019. doi:10.1137/1.9781611975482.81.
- 14 Vijay Kumar and Eric J. Schwabe. Improved Algorithms and Data Structures for Solving Graph Problems in External Memory. In *Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing (SPDP '96)*, SPDP '96, pages 169–, Washington, DC, USA, 1996. IEEE Computer Society. URL: <http://dl.acm.org/citation.cfm?id=829517.830723>.
- 15 Jeffrey Scott Vitter. External Memory Algorithms and Data Structures: Dealing with Massive Data. *ACM Comput. Surv.*, 33(2):209–271, June 2001. doi:10.1145/384192.384193.
- 16 Zhewei Wei and Ke Yi. Equivalence between Priority Queues and Sorting in External Memory. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014*, pages 830–841, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

Shortest Reconfiguration of Perfect Matchings via Alternating Cycles

Takehiro Ito 

Tohoku University, Sendai, Japan
takehiro@ecei.tohoku.ac.jp

Naonori Kakimura

Keio University, Yokohama, Japan
kakimura@math.keio.ac.jp

Naoyuki Kamiyama

Kyushu University, Fukuoka, Japan
JST, PRESTO, Kawaguchi, Japan
kamiyama@imi.kyushu-u.ac.jp

Yusuke Kobayashi

Kyoto University, Kyoto, Japan
yusuke@kurims.kyoto-u.ac.jp

Yoshio Okamoto 

University of Electro-Communications, Chofu, Japan
RIKEN Center for Advanced Intelligence Project, Tokyo, Japan
okamotoy@uec.ac.jp

Abstract

Motivated by adjacency in perfect matching polytopes, we study the shortest reconfiguration problem of perfect matchings via alternating cycles. Namely, we want to find a shortest sequence of perfect matchings which transforms one given perfect matching to another given perfect matching such that the symmetric difference of each pair of consecutive perfect matchings is a single cycle. The problem is equivalent to the combinatorial shortest path problem in perfect matching polytopes. We prove that the problem is NP-hard even when a given graph is planar or bipartite, but it can be solved in polynomial time when the graph is outerplanar.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Matching, Combinatorial reconfiguration, Alternating cycles, Combinatorial shortest paths

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.61

Related Version <https://arxiv.org/abs/1907.01700>

Funding *Takehiro Ito*: Partially supported by JST CREST Grant Number JPMJCR1402, and JSPS KAKENHI Grant Numbers JP18H04091 and JP19K11814, Japan.

Naonori Kakimura: Partially supported by JSPS KAKENHI Grant Numbers JP17K00028 and JP18H05291, Japan.

Naoyuki Kamiyama: Partially supported by JST PRESTO Grant Number JPMJPR1753, Japan.

Yusuke Kobayashi: Partly supported by JSPS KAKENHI Grant Numbers JP16K16010, JP17K19960, and JP18H05291, Japan.

Yoshio Okamoto: Partially supported by JSPS KAKENHI Grant Number 15K00009 and JST CREST Grant Number JPMJCR1402, and Kayamori Foundation of Informational Science Advancement.



© Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto; licensed under Creative Commons License CC-BY

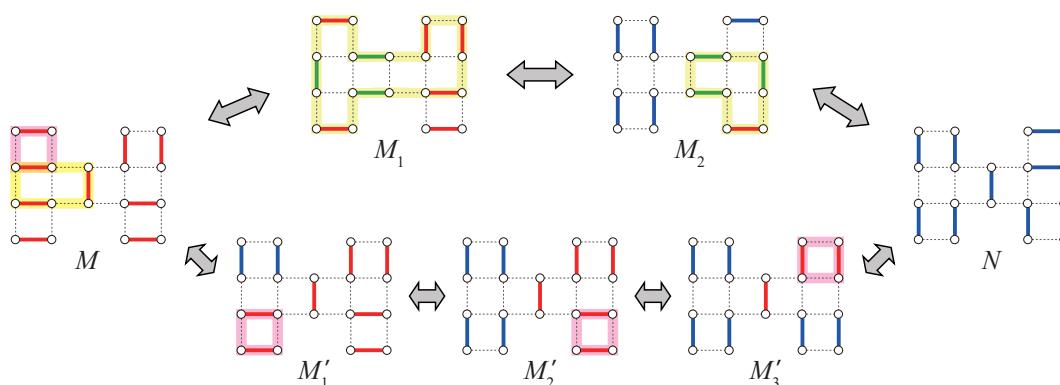
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 61; pp. 61:1–61:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Two sequences of perfect matchings between M and N under the alternating cycle model. The sequence $\langle M, M_1, M_2, N \rangle$ following the yellow alternating cycles is shortest even though it touches the edge in $M \cap N$ twice. On the other hand, $\langle M, M'_1, M'_2, M'_3, N \rangle$ following the pink alternating cycles is not shortest although it touches only the edges in $M \Delta N$.

1 Introduction

Combinatorial reconfiguration is a fundamental research subject that sheds light on solution spaces of combinatorial (search) problems, and connects various concepts such as optimization, counting, enumeration, and sampling. In its general form, combinatorial reconfiguration is concerned with properties of the configuration space of a combinatorial problem. The configuration space of a combinatorial problem is often represented as a graph, but its size is usually exponential in the instance size. Thus, algorithmic problems on combinatorial reconfiguration are not trivial, and require novel tools for resolution. For recent surveys, see [11, 7].

Two basic questions have been encountered in the study of combinatorial reconfiguration. The first question asks the existence of a path between two given solutions in the configuration space, namely the *reachability* of the two solutions. The second question asks the shortest length of a path between two given solutions, if it exists. The second question is usually referred to as a *shortest reconfiguration problem*.

In this paper, we focus on reconfiguration problems of matchings, namely sets of independent edges. There are several ways of defining the configuration space for matchings, and some of them have already been studied in the literature [8, 9, 6, 3, 2]. We will explain them in Section 1.1.

We study yet another configuration space for matchings, which we call the *alternating path/cycle* model. The model is motivated by adjacency in matching polytopes, which we will see soon. In the model, we are given an undirected and unweighted graph G , and also an integer $k \geq 0$. The vertex set of the configuration space consists of the matchings in G of size at least k . Two matchings M and N in G are adjacent in the configuration space if and only if their symmetric difference $M \Delta N := (M \cup N) \setminus (M \cap N)$ is a single path or cycle. In particular, we are interested in the case where $k = |V(G)|/2$, namely the reconfiguration of *perfect matchings*. In that case, the model is simplified to the *alternating cycle* model since $M \Delta N$ cannot have a path. See Figure 1 as an example.

The reachability of two perfect matchings is trivial under the alternating cycle model: the answer is always yes. This is because the symmetric difference of two perfect matchings always consists of vertex-disjoint cycles. Therefore, we focus on the shortest perfect matching reconfiguration under the alternating cycle model.

1.1 Related Work¹

Other Configuration Spaces for Matchings

As mentioned, reconfiguration problems of matchings have already been studied under different models [8, 9, 6, 3, 2]. These models chose more elementary changes as the adjacency on the configuration space. Then, the situation changes drastically under such models: even the reachability of two matchings is not guaranteed.

Matching reconfiguration was initiated by the work of Ito et al. [8]. They proposed the *token addition/removal* model of reconfiguration, in which we are also given an integer $k \geq 0$, and the vertex set of the configuration space consists of the matchings of size at least k .² Two matchings M and N are adjacent if and only if they differ in only one edge. Ito et al. [8] proved that the reachability of two given matchings can be checked in polynomial time.

Another model of reconfiguration is *token jumping*, introduced by Kamiński et al. [9]. In the token jumping model, we are also given an integer $k \geq 0$, and the vertex set of the configuration space consists of the matchings of size exactly k . Two matchings M and N are adjacent if and only if they differ in only two edges. Kamiński et al. [9, Theorem 1] proved that the token jumping model is equivalent to the token addition/removal model when two given matchings have the same size. Thus, using the result by Ito et al. [8], the reachability can be checked in polynomial time also under the token jumping model [9, Corollary 2].

On the other hand, the shortest matching reconfiguration is known to be hard. Gupta et al. [6] and Bousquet et al. [3] independently proved that the problem is NP-hard under the token jumping model. Then, the problem is also NP-hard under the token addition/removal model, because the shortest lengths are preserved under the two models [9, Theorem 1].

Recently, Bonamy et al. [2] studied the reachability of two perfect matchings under a model close to ours, namely the alternating cycle model *restricted to length four*. In the model, two perfect matchings M and N are adjacent if and only if their symmetric difference $M \triangle N$ is a cycle of length four. Then, the answer to the reachability is not always yes, and Bonamy et al. [2] proved that the reachability problem is PSPACE-complete under this restricted model.

Relation to Matching Polytopes

Our alternating cycle model (without any restriction of cycle length) for the perfect matching reconfiguration is natural when we see the connection with the simplex methods for linear optimization, or combinatorial shortest paths of the graphs of convex polytopes.

In the combinatorial shortest path problem of a convex polytope, we are given a convex polytope P , explicitly or implicitly, and two vertices v, w of P . Then, we want to find a shortest sequence u_0, u_1, \dots, u_t of vertices of P such that $u_0 = v, u_t = w$ and $\overline{u_i u_{i+1}}$ forms an edge of P for every $i = 0, 1, \dots, t - 1$. Often, we are only interested in the length of such a shortest sequence, and we are also interested in the maximum shortest path length among all pairs of vertices, which is known as the combinatorial diameter of the polytope P . The combinatorial diameter of a polytope has attracted much attention in the optimization community from the motivation of better understanding of simplex methods. Simplex methods for linear optimization start at a vertex of the feasible region, follow edges, and arrive at an optimal vertex. Therefore, the combinatorial diameter dictates the best-case

¹ Further related work can be found in the full version.

² Precisely, their model is defined in a slightly different way, but it is essentially the same as this definition.

behavior of such methods. The famous Hirsch conjecture states that every d -dimensional convex polytope with n facets has the combinatorial diameter at most $n - d$. This has been disproved by Santos [14], and the current best upper bound of $(n - d)^{\log_2 O(d/\log d)}$ for the combinatorial diameter was given by Sukegawa [15]. On the other hand, for the 0/1-polytopes (i.e., polytopes in which the coordinates of all vertices belong to $\{0, 1\}$), the Hirsch conjecture holds [10].

The shortest perfect matching reconfiguration under the alternating cycle model can be seen as the combinatorial shortest path problem of a perfect matching polytope. The *perfect matching polytope* of a graph G is defined as follows. The polytope lives in $\mathbb{R}^{E(G)}$, namely each coordinate corresponds to an edge of G . Each vertex v of the polytope corresponds to a perfect matching M of G as $v_e = 1$ if $e \in M$ and $v_e = 0$ if $e \notin M$. The polytope is defined as the convex hull of those vertices. It is known that two vertices u, v of the perfect matching polytope form an edge if and only if the corresponding perfect matchings M, N have the property that $M \triangle N$ contains only one cycle [4]. This means that the graph of the perfect matching polytope is exactly the configuration space for perfect matchings under the alternating cycle model.

1.2 Our Contribution

To the best of the authors' knowledge, known results under different models do not have direct relations to our alternating cycle model, because their configuration spaces are different. In this paper, we thus investigate the polynomial-time solvability of the shortest perfect matching reconfiguration under the alternating cycle model. The results of our paper are two-fold.

1. The shortest perfect matching reconfiguration under the alternating cycle model can be solved in polynomial time if the input graph is outerplanar.
2. The shortest perfect matching reconfiguration under the alternating cycle model is NP-hard even when the input graph is planar or bipartite.

Since outerplanar graphs form a natural and fundamental subclass of planar graphs, our results exhibit a tractability border among planar graphs.

The hardness result for bipartite graphs implies that the computation of a combinatorial shortest path in a convex polytope is NP-hard even when an inequality description is explicitly given. This is because a polynomial-size inequality description of the perfect matching polytope can be explicitly written down from a given bipartite graph.

We point out that the hardness results have been independently obtained by Aichholzer et al. [1]. Indeed, they proved the hardness for planar bipartite graphs (i.e., an input graph is planar *and* bipartite).

Technical Key Points

Compared to recent algorithmic developments on reachability problems, only a few polynomial-time solvable cases are known for shortest reconfiguration problems. We now explain two technical key points, especially for algorithmic results on shortest reconfiguration problems.

The first point is the symmetric difference of two given solutions. Under several known models (not only for matchings) that employ elementary changes as the adjacency on the configuration space, the symmetric difference gives a (good) lower bound on the shortest reconfiguration. This is because any reconfiguration sequence (i.e., a path in the configuration space) between two given solutions must touch all elements in their symmetric difference at least once. For example, in Figure 1, the symmetric difference of two perfect matchings M and N consists of 16 edges and hence it gives the lower bound of $16/4 = 4$ under

the alternating cycle model restricted to length 4 [2]. In such a case, if we can find a reconfiguration sequence touching only the elements in the symmetric difference (e.g., the sequence $\langle M, M'_1, M'_2, M'_3, N \rangle$ in Figure 1), then it is automatically the shortest under that model. However, this useful property does not hold under our alternating cycle model, because the length of an alternating cycle for reconfiguration is not fixed.

The second point is the characterization of *unhappy moves* that touch elements contained commonly in two given solutions. For example, the shortest reconfiguration sequence $\langle M, M_1, M_2, N \rangle$ in Figure 1 has an unhappy move, since it touches the edge in $M \cap N$ twice. In general, analyzing a shortest reconfiguration becomes much more difficult if such unhappy moves are required. A well-known example is the (generalized) 15-puzzle [13] in which the reachability can be determined in polynomial time, while the shortest reconfiguration is NP-hard. As illustrated in Figure 1, the shortest perfect matching reconfiguration requires unhappy moves even for outerplanar graphs, and hence we need to characterize the unhappy moves to develop a polynomial-time algorithm.

2 Problem Definition

In this paper, a graph always refers to an undirected graph that might have parallel edges and does not have loops. For a graph G , we denote by $V(G)$ and $E(G)$ the vertex set and edge set of G , respectively. An edge subset $M \subseteq E$ is called a *matching* in G if no two edges in M share the end vertices. A matching M is *perfect* if $|M| = |V(G)|/2$.

A graph is *planar* if it can be drawn on the plane without edge crossing. Such a drawing is called a *plane* drawing of the planar graph. A *face* of a plane drawing is a maximal region of the plane that contains no point used in the drawing. There is a unique unbounded face, which is called the *outer face*. A planar graph is *outerplanar* if it has an *outerplane* drawing, i.e., a plane drawing in which all vertices are incident to the outer face.

For a matching M in a graph G , a cycle C in G is called *M -alternating* if edges in M and $E(G) \setminus M$ alternate in C . We identify a cycle with its edge set to simplify the notation. We say that two perfect matchings M and N are *reachable* (under the alternating cycle model) if there exists a sequence $\langle M_0, M_1, \dots, M_t \rangle$ of perfect matchings in G such that

- (i) $M_0 = M$ and $M_t = N$; and
- (ii) $M_i = M_{i-1} \triangle C_i$ for some M_{i-1} -alternating cycle C_i for each $i = 1, \dots, t$.

Such a sequence is called a *reconfiguration sequence* between M and N , and its *length* is defined as t .

For two perfect matchings M and N , the subgraph $M \triangle N$ consists of disjoint M -alternating cycles C_1, \dots, C_t . Thus it is clear that M and N are always reachable for any two perfect matchings M and N by setting $M_i = M_{i-1} \triangle C_i$ for $i = 1, \dots, t$. In this paper, we are interested in finding a *shortest* reconfiguration sequence of perfect matchings. That is, the problem is defined as follows:

SHORTEST PERFECT MATCHING RECONFIGURATION

Input: A graph G and two perfect matchings M and N in G

Find: A shortest reconfiguration sequence between M and N .

We denote by a tuple $I = (G, M, N)$ an instance of SHORTEST PERFECT MATCHING RECONFIGURATION. Also, we denote by $\text{OPT}(I)$ the length of a shortest reconfiguration sequence of an instance I . We note that it may happen that $\text{OPT}(I)$ is much shorter than the number of disjoint M -alternating cycles in $M \triangle N$ (see Figure 1).

3 Polynomial-Time Algorithm for Outerplanar Graphs

In this section, we prove that there exists a polynomial-time algorithm for SHORTEST PERFECT MATCHING RECONFIGURATION on an outerplanar graph, as follows.

► **Theorem 1.** SHORTEST PERFECT MATCHING RECONFIGURATION *on outerplanar graphs* G can be solved in $O(|V(G)|^5)$ time.

We give such an algorithm in this section. Let $I = (G, M, N)$ be an instance of the problem such that $G = (V, E)$ is an outerplanar graph. We first observe that it suffices to consider the case when G is 2-connected.

► **Lemma 2** (*³). *Let $I = (G, M, N)$ be an instance of SHORTEST PERFECT MATCHING RECONFIGURATION, and G_1, \dots, G_p be the 2-connected components of G . Furthermore, for every $i = 1, \dots, p$, let $I_i = (G_i, M \cap E(G_i), N \cap E(G_i))$ be an instance of SHORTEST PERFECT MATCHING RECONFIGURATION. Then, $\text{OPT}(I) = \sum_{i=1}^p \text{OPT}(I_i)$.*

Since the 2-connected components of a graph can be found in linear time, the reduction to 2-connected outerplanar graphs can be done in linear time, too.

We fix an outerplane drawing of a given 2-connected outerplanar graph G , and identify G with the drawing for the sake of convenience. We denote by C_{out} the outer face boundary. Then C_{out} is a simple cycle since G is 2-connected. We denote the set of the *inner edges* of G by $E_{\text{in}} = E \setminus C_{\text{out}}$. In other words, E_{in} is the set of chords of C_{out} .

3.1 Technical Highlight

As mentioned in Introduction, there are two technical key points to develop a polynomial-time algorithm for SHORTEST PERFECT MATCHING RECONFIGURATION: a lower bound on the length of a shortest reconfiguration sequence, and the characterization of unhappy moves. We here explain our ideas roughly, and will give detailed descriptions in the next subsections.

Since G is planar, we can define its “dual-like” graph G^* . Then, G^* forms a tree since G is outerplanar and 2-connected. (The definition of G^* will be given in Section 3.2, and an example is given in Figure 2.) We make a correspondence between an edge in G^* and a set of edges in G . Then, we will define the length $\ell(e^*)$ of each edge e^* in G^* so that it represents the “gap” between M and N when we are restricted to the edges in the corresponding set of e^* . It is important to notice that any cycle C in G corresponds to a subtree of G^* , and vice versa. Indeed, we focus on a cut C^* of G^* clipping the subtree from G^* , that is, the set of edges in G^* leaving the subtree. If we apply an M -alternating cycle C to a perfect matching M of G , then it changes lengths $\ell(e^*)$ of the edges e^* in the corresponding cut C^* .

For our algorithm, we need a (good) lower bound for the length of a shortest reconfiguration sequence between two given perfect matchings M and N . Recall that $|M \triangle N|$ does not give a good lower bound under the alternating cycle model. This is because we can take a cycle of an arbitrary (non-fixed) length, and hence $|M \triangle N|$ can decrease drastically by only a single alternating cycle. Furthermore, no matter how we define the length $\ell(e^*)$ of each edge e^* in G^* , the total length of *all* edges in G^* does not give a good lower bound. This is because a cycle C of non-fixed length in G may correspond to a cut C^* having many edges in G^* , and hence it can change the total length drastically. Our key idea is to focus on the total length of each *path* in G^* , that is, we take the *diameter* of G^* (with respect to length ℓ)

³ The symbol (*) means that the proof is postponed to the full version.

as a lower bound. Then, because G^* is a tree, any path in G^* can contain at most two edges from the corresponding cut C^* . Therefore, regardless of the cycle length, the diameter of G^* can be changed by only these two edges. By carefully setting the length $\ell(e^*)$ as in (1), we will prove that the diameter of G^* is not only a lower bound, but indeed gives the shortest length under the assumption that $E_{\text{in}} \cap M \cap N$ is empty. Therefore, the real difficulty arises when $E_{\text{in}} \cap M \cap N$ is not empty.

In the latter case, we will characterize the unhappy moves. Assume that we know the set $F \subseteq E_{\text{in}} \cap M \cap N$ of chords that are *not* touched in a shortest reconfiguration sequence between M and N ; in other words, *all* chords in $(E_{\text{in}} \cap M \cap N) \setminus F$ must be touched for unhappy moves in that sequence. Then, we subdivide a given outerplanar graph G into subgraphs $G_1, \dots, G_{|F|+1}$ along the chords in F . Notice that each edge in F appears on the outer face boundaries in two of these subgraphs. Furthermore, each chord e in these subgraphs satisfies $e \in (E_{\text{in}} \cap M \cap N) \setminus F$ if $e \in M \cap N$. Therefore, *all* chords in these subgraphs are touched for unhappy moves as long as they are in $M \cap N$. Under this assumption, we will prove that the diameter of G_i^* gives the shortest length of a reconfiguration sequence between $M \cap E(G_i)$ and $N \cap E(G_i)$. Thus, we can solve the problem in polynomial time if we know F which yields a shortest reconfiguration sequence between M and N . Finally, to find such a set F of chords, we construct a polynomial-time algorithm which employs a dynamic programming method along the tree G^* .

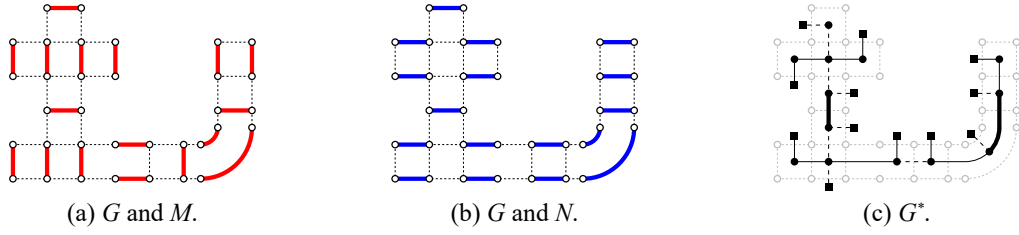
3.2 Preliminaries: Constructing a Dual Graph

Let $I = (G, M, N)$ be an instance of SHORTEST PERFECT MATCHING RECONFIGURATION such that G is a 2-connected outerplanar graph. Since G is planar, we can define the *dual* of G . In fact, we here construct a graph G^* obtained from the dual by applying a slight modification as follows. The construction is illustrated in Figure 2. Let V^* be the set of faces (without the outer face) of G . For a face $v^* \in V^*$, let $E_{v^*} \subseteq E(G)$ be the set of edges around v^* . We denote the set of faces touching the outer face by U^* , i.e., $U^* = \{v^* \in V^* \mid E_{v^*} \cap C_{\text{out}} \neq \emptyset\}$. We make a copy of U^* , denoted by \tilde{U}^* . We set the vertex set of G^* to be $V^* \cup \tilde{U}^*$. For v^*, w^* in V^* , an edge v^*w^* in G^* exists if and only if the faces v^* and w^* share an edge in E_{in} , i.e., $|E_{v^*} \cap E_{w^*}| = 1$. Also G^* has an edge between u^* and \tilde{u}^* for every $u^* \in U^*$, where $\tilde{u}^* \in \tilde{U}^*$ is the copy of u^* . Thus the edge set of G^* is given by

$$E(G^*) = \{v^*w^* \mid v^*, w^* \in V^*, |E_{v^*} \cap E_{w^*}| = 1\} \cup \{u^*\tilde{u}^* \mid u^* \in U^*\}.$$

The first part is denoted by E_{in}^* , and the second part is denoted by \tilde{E}^* . We observe that G^* is a tree, since G is 2-connected and outerplanar. A face of G that touches only one face (other than the outer face) is called a *leaf* in $G^* - \tilde{U}^*$. We note that there is a one-to-one correspondence between edges in E_{in} of G and E_{in}^* of G^* . For an edge subset $F \subseteq E_{\text{in}}$, F^* denotes the corresponding edge subset in G^* , that is, $F^* = \{e^* \in E_{\text{in}}^* \mid e \in F\}$. Conversely, for an edge subset $F^* \subseteq E(G^*)$, F denotes the corresponding edge subset in E_{in} , that is, $F = \{e \in E_{\text{in}} \mid e^* \in F^* \cap E_{\text{in}}^*\}$. We extend this correspondence to \tilde{E}^* , that is, $u^*\tilde{u}^* \in \tilde{E}^*$ corresponds to the edge set $E_{u^*} \cap C_{\text{out}}$ for $u^* \in U^*$, and vice versa.

It follows from the duality that there is a relationship between a cut in G^* and a cycle in G . Suppose that we are given a cycle C ($\neq C_{\text{out}}$) in G . Then, since G is outerplanar, the cycle C surrounds a set X^* of faces such that X^* does not have the outer face. The set X^* induces a connected graph (subtree) in G^* , and the set of edges leaving from X^* yields a cut $C^* = \{e^* = v^*w^* \mid v^* \in X^*, w^* \in V(G^*) \setminus X^*\}$. Conversely, let $X^* \subseteq V^*$ be a vertex subset of G^* such that the subgraph induced by X^* is connected. Then the set of edges leaving from X^* yields a cut C^* in G^* , which corresponds to a cycle in G .



■ **Figure 2** The construction of G^* and the length function ℓ . In (c), the edge lengths are depicted by different styles: thick solid lines represent edges of length two, thin solid lines represent edges of length one, and dotted lines represent edges of length zero.

We classify faces in U^* into two groups. For a face u^* in U^* , the edge set $E_{u^*} \cap C_{\text{out}}$ forms a family \mathcal{P}_{u^*} of disjoint paths. Since M and N are perfect matchings, each path P in \mathcal{P}_{u^*} is both M -alternating and N -alternating. In addition, P satisfies either

- (i) $E(P) \subseteq M \triangle N$, or
- (ii) $(M \triangle N) \cap E(P) = \emptyset$.

Furthermore, we observe that either (i) holds for *every* path P in \mathcal{P}_{u^*} , or (ii) holds for *every* path P in \mathcal{P}_{u^*} . Indeed, since $M \triangle N$ consists of disjoint cycles, if some path P in \mathcal{P}_{u^*} satisfies (i), then P is included in a cycle C in $M \triangle N$ that separates u^* from the outer face. Since the other paths in \mathcal{P}_{u^*} touch the outer face, they are on C . Thus every path satisfies (i), which shows the observation. We divide U^* into two groups U_1^* and U_2^* where each face in U_1^* satisfies (i) for every path, while each face in U_2^* satisfies (ii) for every path.

For an edge e^* in $E(G^*)$, we define the length $\ell(e^*)$ to be

$$\ell(e^*) = \begin{cases} |M \cap \{e\}| + |N \cap \{e\}| & \text{if } e^* \in E_{\text{in}}^*; \\ 1 & \text{if } e^* = u^* \tilde{u}^* \in \tilde{E}^* \text{ such that } u^* \in U_1^*; \\ 0 & \text{if } e^* = u^* \tilde{u}^* \in \tilde{E}^* \text{ such that } u^* \in U_2^*. \end{cases} \quad (1)$$

See Figure 2 for an example. Let $\ell(u^*, v^*)$ be the length of the (unique) path from u^* to v^* in G^* . We define the *gap* between M and N in the graph G as the diameter of G^* , that is, we define $\text{gap}(I) = \max\{\ell(u^*, v^*) \mid u^*, v^* \in V(G^*)\}$. This value is simply denoted by $\text{gap}(M, N)$ if G is clear from the context.

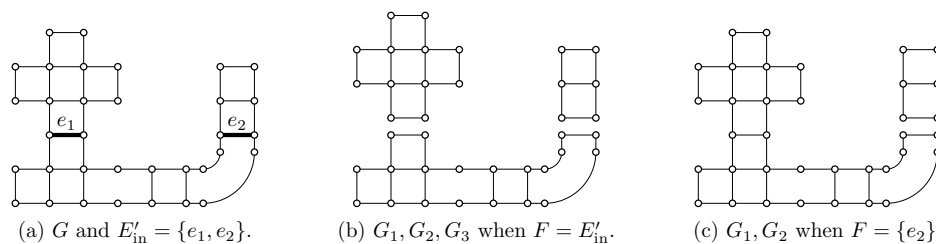
3.3 Characterization for the Disjoint Case

Let $I = (G, M, N)$ be an instance of SHORTEST PERFECT MATCHING RECONFIGURATION such that G is a 2-connected outerplanar graph. In this subsection, we show that if $E_{\text{in}} \cap M \cap N$ is empty, we can characterize the optimal value with $\text{gap}(I)$, which leads to a simple polynomial-time algorithm for this case. We note that if $E_{\text{in}} \cap M \cap N$ is empty, then no edge in E_{in} belongs to both M and N , and hence $\ell(e^*)$ can only take 0 or 1.

► **Lemma 3** (*). *It holds that $\text{gap}(M, N)$ is even.*

A main theorem of this subsection is to give a characterization of the optimal value with $\text{gap}(M, N)$.

► **Theorem 4.** *Let $I = (G, M, N)$ be an instance of SHORTEST PERFECT MATCHING RECONFIGURATION such that G is a 2-connected outerplanar graph. If $E_{\text{in}} \cap M \cap N$ is empty, then it holds that $\text{OPT}(I) = \text{gap}(M, N)/2$.*



■ **Figure 3** Decomposition of the outerplanar graph in Figure 2. The edges in E'_{in} are shown with bold lines.

Proof. To show the theorem, we first prove the following claim.

▷ **Claim 5 (*)**. For any M -alternating cycle C , it holds that $\text{gap}(M, N) \leq \text{gap}(M \triangle C, N) + 2$.

Consider a shortest reconfiguration sequence $\langle M_0, M_1, \dots, M_t \rangle$ from $M_0 = M$ to $M_t = N$. Then, $t = \text{OPT}(I)$. For each $i = 1, \dots, t$, it then holds that $\text{gap}(M_{i-1}, N) \leq \text{gap}(M_i, N) + 2$. By repeatedly applying the above inequalities, we obtain

$$\text{gap}(M, N) = \text{gap}(M_0, N) \leq \text{gap}(M_t, N) + 2t = 2t = 2\text{OPT}(I)$$

since $\text{gap}(M_t, N) = 0$. Hence it holds that $\text{OPT}(I) \geq \text{gap}(M, N)/2$.

It remains to show that $\text{OPT}(I) \leq \text{gap}(M, N)/2$. We prove the following claim.

▷ **Claim 6 (*)**. There exists an M -alternating cycle C such that $\text{gap}(M, N) = \text{gap}(M \triangle C, N) + 2$.

For a perfect matching M_{i-1} in G , it follows from Claim 6 that there exists an M_{i-1} -alternating cycle C_i such that $\text{gap}(M_{i-1}, N) = \text{gap}(M_{i-1} \triangle C_i, N) + 2$. Define $M_i = M_{i-1} \triangle C_i$, and repeat finding an alternating cycle satisfying the above equation. The repetition ends when $\text{gap}(M_i, N) = 0$, which means that $M_i = N$. The number of repetitions is equal to $\text{gap}(M, N)/2$, and therefore, we have $\text{OPT}(I) \leq \text{gap}(M, N)/2$. Thus the proof is complete. ◀

3.4 General Case

Let $I = (G, M, N)$ be an instance of SHORTEST PERFECT MATCHING RECONFIGURATION such that G is a 2-connected outerplanar graph. Define $E'_{\text{in}} = E_{\text{in}} \cap M \cap N$. In this subsection, we deal with the general case, that is, E'_{in} is not necessarily empty. Then, there is a case when changing an edge in E'_{in} reduces the number of reconfiguration steps as in Figure 1. We call such a move an *unhappy move*. The key idea of our algorithm is to detect a set of edges necessary for unhappy moves.

Since G is outerplanar and 2-connected, any $F \subseteq E'_{\text{in}}$ divides the inner region of C_{out} into $|F| + 1$ parts $R_1, \dots, R_{|F|+1}$. For each $i = 1, \dots, |F| + 1$, let G_i be the subgraph of G consisting of all the vertices and the edges in R_i and its boundary. Thus, each edge $e \in F$ appears on the outer face boundaries in two of these subgraphs. See Figure 3. Let $\mathcal{G}_F = \{G_1, \dots, G_{|F|+1}\}$. Note that each graph in \mathcal{G}_F is outerplanar and 2-connected. For each $H \in \mathcal{G}_F$, let $I_H = (H, M \cap E(H), N \cap E(H))$. We now show the following theorem.

► **Theorem 7.** $\text{OPT}(I) = \frac{1}{2} \min_{F \subseteq E'_{\text{in}}} \sum_{H \in \mathcal{G}_F} \text{gap}(I_H)$.

61:10 Shortest Reconfiguration of Perfect Matchings via Alternating Cycles

Proof. Let $\langle M_0, M_1, \dots, M_t \rangle$ be a shortest reconfiguration sequence from $M_0 = M$ to $M_t = N$. We denote by C_i the M_{i-1} -alternating cycle with $M_i = M_{i-1} \Delta C_i$. Define $F_{\text{opt}} = \{e \in E'_{\text{in}} \mid e \notin C_i, \forall i\}$, which is the set of edges in E'_{in} that are not touched in the shortest reconfiguration sequence. Then C_i is contained in some $H \in \mathcal{G}_{F_{\text{opt}}}$, and can be used to obtain a reconfiguration sequence from $M \cap E(H)$ to $N \cap E(H)$ in H . Therefore, we have

$$\text{OPT}(I) = \sum_{H \in \mathcal{G}_{F_{\text{opt}}}} \text{OPT}(I_H). \quad (2)$$

We can also see that

$$\text{OPT}(I) \leq \sum_{H \in \mathcal{G}_F} \text{OPT}(I_H) \quad (3)$$

for any $F \subseteq E'_{\text{in}}$.

To evaluate $\text{OPT}(I_H)$ for $H \in \mathcal{G}_F$, we slightly modify the instance I_H by replacing every inner edge of H contained in $M \cap N$ by two parallel edges each in M and N , respectively. The obtained graph is denoted by H' , and the corresponding instance is denoted by $I_{H'}$. Since a reconfiguration sequence for $I_{H'}$ can be converted to one for I_H , it holds that $\text{OPT}(I_H) \leq \text{OPT}(I_{H'})$, and hence

$$\text{OPT}(I) \leq \sum_{H \in \mathcal{G}_F} \text{OPT}(I_H) \leq \sum_{H \in \mathcal{G}_F} \text{OPT}(I_{H'}) \quad (4)$$

holds for any $F \subseteq E'_{\text{in}}$ by (3). Moreover, by the definition of F_{opt} , there exists an index i such that $e \in C_i$ for any $e \in E'_{\text{in}} \setminus F_{\text{opt}}$. Therefore, for $H \in \mathcal{G}_{F_{\text{opt}}}$, the shortest reconfiguration sequence for I_H can be converted to a reconfiguration sequence for $I_{H'}$. Thus, $\text{OPT}(I_H) \geq \text{OPT}(I_{H'})$ holds for $H \in \mathcal{G}_{F_{\text{opt}}}$, and hence

$$\text{OPT}(I) = \sum_{H \in \mathcal{G}_{F_{\text{opt}}}} \text{OPT}(I_H) \geq \sum_{H \in \mathcal{G}_{F_{\text{opt}}}} \text{OPT}(I_{H'}) \quad (5)$$

by (2). By (4) and (5), we obtain

$$\text{OPT}(I) = \min_{F \subseteq E'_{\text{in}}} \sum_{H \in \mathcal{G}_F} \text{OPT}(I_{H'}), \quad (6)$$

and F_{opt} is a minimizer of the right-hand side.

By (6) and Theorem 4, we obtain

$$\text{OPT}(I) = \frac{1}{2} \min_{F \subseteq E'_{\text{in}}} \sum_{H \in \mathcal{G}_F} \text{gap}(I_{H'}), \quad (7)$$

because each $I_{H'}$ satisfies the condition in Theorem 4. Since $(H')^*$ is obtained from H^* by subdividing some edges of length two into two edges of length one, the diameter of $(H')^*$ is equal to that of H^* , that is, $\text{gap}(I_{H'}) = \text{gap}(I_H)$. Therefore, we obtain the theorem by (7). ◀

As an example, we apply this theorem to the instance in Figure 2. See Figure 3(c). If F consists of only the right thick edge in Figure 2(c), then \mathcal{G}_F consists two graphs G_1 and G_2 such that $\text{gap}(I_{G_1}) = 6$ and $\text{gap}(I_{G_2}) = 2$. Since we can check that such F attains the minimum in the right-hand side of Theorem 7, we obtain $\text{OPT}(I) = 4$ by Theorem 7.

In order to compute the value in Theorem 7 efficiently, we reduce the problem to MIN-SUM DIAMETER DECOMPOSITION, whose definition will be given later.

For $F \subseteq E'_{\text{in}}$, let F^* be the edge subset of E_{in}^* corresponding to F , and let $\mathcal{G}_F = \{G_1, \dots, G_{|F|+1}\}$. Then, $G^* - F^*$ consists of $|F| + 1$ components $T_1, T_2, \dots, T_{|F|+1}$ such that T_i coincides with G_i^* (except for the difference of edges of length zero) for $i = 1, \dots, |F| + 1$. In particular, for each i , we have $\text{gap}(I_{G_i}) = \max\{\ell(u^*, v^*) \mid u^*, v^* \in V(T_i)\}$, where ℓ is the length function on $E(G^*)$ defined by the instance $I = (G, M, N)$. We call $\max\{\ell(u^*, v^*) \mid u^*, v^* \in V(T_i)\}$ the *diameter* of T_i , which is denoted by $\text{diam}_\ell(T_i)$. Then, Theorem 7 shows that

$$\text{OPT}(I) = \frac{1}{2} \min_{F \subseteq E'_{\text{in}}} \sum_{i=1}^{|F|+1} \text{diam}_\ell(T_i). \quad (8)$$

Therefore, we can compute $\text{OPT}(I)$ by solving the following problem in which $T = G^*$ and $E_0 = (E'_{\text{in}})^*$.

MIN-SUM DIAMETER DECOMPOSITION

Input: A tree T , an edge subset $E_0 \subseteq E(T)$, and a length function $\ell : E(T) \rightarrow \mathbb{Z}_{\geq 0}$

Find: An edge set $F \subseteq E_0$ that minimizes $\sum_{T'} \text{diam}_\ell(T')$, where the sum is taken over all the components T' of $T - F$.

In the subsequent subsection, we show that MIN-SUM DIAMETER DECOMPOSITION can be solved in time polynomial in $|V(T)|$ and $L := \sum_{e \in E(T)} \ell(e)$.

► **Theorem 8.** MIN-SUM DIAMETER DECOMPOSITION can be solved in $O(|V(T)|L^4)$ time, where $L := \sum_{e \in E(T)} \ell(e)$.

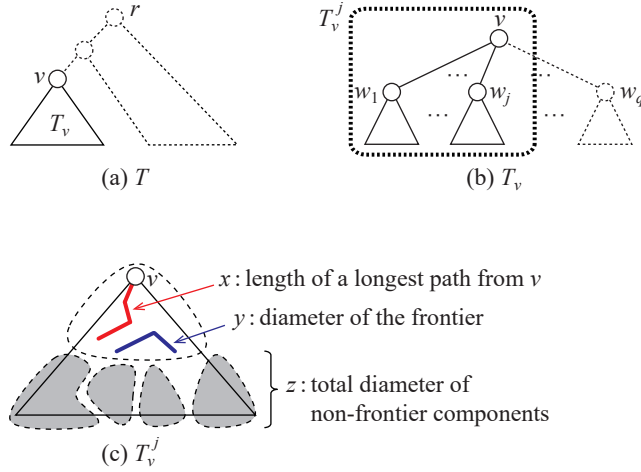
Since (8) shows that SHORTEST PERFECT MATCHING RECONFIGURATION on outerplanar graphs is reduced to MIN-SUM DIAMETER DECOMPOSITION in which $L = O(|V(T)|)$, we obtain Theorem 1.

3.5 Algorithm for Min-Sum Diameter Decomposition

The remaining task is to show Theorem 8, that is, to give an algorithm for MIN-SUM DIAMETER DECOMPOSITION that runs in $O(|V(T)|L^4)$ time. For this purpose, we adopt a dynamic programming approach.

We choose an arbitrary vertex r of a given tree T , and regard T as a rooted tree with the root r . For each vertex v of T , we denote by T_v the subtree of T which is rooted at v and is induced by all descendants of v in T . (See Figure 4(a).) Thus, $T = T_r$ for the root r . Let w_1, w_2, \dots, w_q be the children of v , ordered arbitrarily. For each $j \in \{1, 2, \dots, q\}$, we denote by T_v^j the subtree of T induced by $\{v\} \cup V(T_{w_1}) \cup V(T_{w_2}) \cup \dots \cup V(T_{w_j})$. For example, in Figure 4(b), the subtree T_v^j is surrounded by a thick dotted rectangle. For notational convenience, we denote by T_v^0 the tree consisting of a single vertex v . Then, $T_v = T_v^0$ for each leaf v of T . Our algorithm computes and extends partial solutions for subtrees T_v^j from the leaves to the root r of T by keeping the information required for computing (the sum of) diameters of a partial solution.

We now define partial solutions for subtrees. For a subtree T_v^j and an edge subset $F' \subseteq E_0 \cap E(T_v^j)$, the *frontier* for F' is the component (subtree) in $T_v^j - F'$ that contains the root v of T_v^j . We sometimes call it the *v-frontier* for F' to emphasize the root v . For three integers $x, y, z \in \{0, 1, \dots, L\}$, the edge subset F' is called an (x, y, z) -separator of T_v^j if the following three conditions hold. (See also Figure 4(c).)



■ **Figure 4** (a) Subtree T_v in the whole tree T , (b) subtree T_v^j in T_v , and (c) an (x, y, z) -separator of T_v^j .

- $x = \max\{\ell(v, u) \mid u \in V(T_{F'})\}$, where $T_{F'}$ is the v -frontier for F' . That is, the longest path from v to a vertex in $T_{F'}$ is of length x .
 - $y = \text{diam}_\ell(T_{F'})$, that is, y denotes the diameter of the v -frontier $T_{F'}$ for F' .
 - $z = \sum_{T'} \text{diam}_\ell(T')$, where the sum is taken over all the components T' of $(T - F') \setminus T_{F'}$.
- Note that $x \leq y$ always holds for an (x, y, z) -separator of T_v^j . We then define the following function: for a subtree T_v^j and two integers $x, y \in \{0, 1, \dots, L\}$, we let

$$f(T_v^j; x, y) = \min \{z \mid T_v^j \text{ has an } (x, y, z)\text{-separator}\}.$$

Note that $f(T_v^j; x, y)$ is defined as $+\infty$ if T_v^j does not have an (x, y, z) -separator for any $z \in \{0, 1, \dots, L\}$. Then, the optimal objective value to MIN-SUM DIAMETER DECOMPOSITION can be computed as $\min\{y + f(T; x, y) \mid x, y \in \{0, 1, \dots, L\}\}$.

For a given tree T , our algorithm computes $f(T_v^j; x, y)$ for all possible triplets (T_v^j, x, y) from the leaves to the root r of T . The algorithm runs in $O(|V(T)|L^4)$ time in total. (The details are explained in the full version.) Note that we can easily modify the algorithm so that we obtain not only the optimal value but also an optimal solution. This completes the proof of Theorem 8.

We note here that the algorithm can be modified so that the running time is bounded by a polynomial in $|V(T)|$ by replacing the domain $\{0, 1, \dots, L\}$ of x and y with $D := \{\ell(u, v) \mid u, v \in V(T)\}$. This modification is valid, because $f(T_v^j; x, y) = +\infty$ unless $x, y \in D$. Since $|D| = O(|V(T)|^2)$, the modified algorithm runs in $O(|V(T)||D|^4) = O(|V(T)|^9)$ time. Note that, although this bound is polynomial only in $|V(T)|$, it is worse than $O(|V(T)|L^4)$ when $L = O(|V(T)|)$.

4 NP-Hardness for Planar Graphs and Bipartite Graphs

In this section, we prove that SHORTEST PERFECT MATCHING RECONFIGURATION is NP-hard even when the input graph is planar or bipartite.

► **Theorem 9.** SHORTEST PERFECT MATCHING RECONFIGURATION is NP-hard even for planar graphs of maximum degree three.

We reduce the HAMILTONIAN CYCLE problem, which is known to be NP-complete even when a given graph is 3-regular and planar [5].

HAMILTONIAN CYCLE

Input: A 3-regular planar graph $H = (V, E)$

Question: Decide whether H has a Hamiltonian cycle, i.e., a cycle that goes through all the vertices exactly once.

Proof. Let H be a 3-regular planar graph, which is an instance of HAMILTONIAN CYCLE. For each vertex $v \in V(H)$, we define a 8-vertex graph D_v (see also the top right in Figure 5):

$$\begin{aligned} V(D_v) &= \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}, \\ E(D_v) &= \{v_1v_2, v_2v_3, v_3v_4, v_4v_1, v_4v_5, v_5v_7, v_3v_6, v_6v_8\}. \end{aligned}$$

We construct an instance $I = (G, M, N)$ of our problem as follows. (See Figure 5 as an example.) We subdivide each edge $e = uv$ in H twice, and the obtained vertices are denoted by u_e and v_e , where u_e is closer to u . Then, for each vertex $v \in V(H)$, we replace v with the graph D_v , and connect v_7 to $v_{e_v^{(1)}}$ and $v_{e_v^{(2)}}$, v_8 to $v_{e_v^{(2)}}$ and $v_{e_v^{(3)}}$, where $e_v^{(1)}, e_v^{(2)}, e_v^{(3)}$ are edges incident to v and the order follows the plane drawing of H . Let $E_v = \{v_7v_{e_v^{(1)}}, v_7v_{e_v^{(2)}}, v_8v_{e_v^{(2)}}, v_8v_{e_v^{(3)}}\}$. The resulting graph is denoted by G , i.e., G is defined as follows:

$$\begin{aligned} V(G) &= \bigcup_{v \in V(H)} V(D_v) \cup \bigcup_{e=uv \in E(H)} \{u_e, v_e\}, \\ E(G) &= \left(\bigcup_{v \in V(H)} E(D_v) \cup E_v \right) \cup \{u_e v_e \mid e \in E(H)\}. \end{aligned}$$

It follows that G is a planar graph of maximum degree three. Furthermore, we define initial and target perfect matchings M and N in G , respectively, to be

$$\begin{aligned} M &= \{v_1v_2, v_3v_4, v_5v_7, v_6v_8 \mid v \in V(H)\} \cup \{u_e v_e \mid e \in E(H)\}, \\ N &= \{v_1v_4, v_2v_3, v_5v_7, v_6v_8 \mid v \in V(H)\} \cup \{u_e v_e \mid e \in E(H)\}. \end{aligned}$$

This completes the construction of our corresponding instance $I = (G, M, N)$. The construction can be done in polynomial time.

We then give the following claim.

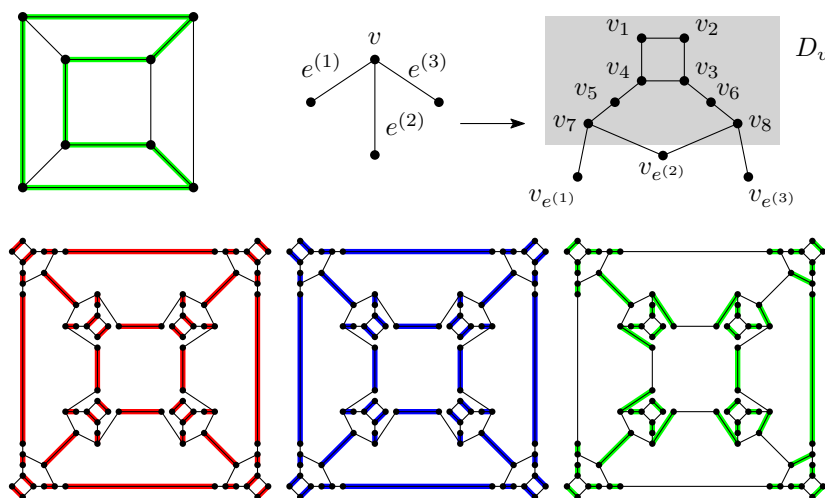
▷ **Claim 10 (*).** H has a Hamiltonian cycle if and only if $\text{OPT}(I) = 2$.

This completes the proof of Theorem 9. ◀

The hardness for bipartite graphs of maximum degree three can be obtained with a similar proof; the reduction uses the DIRECTED HAMILTONIAN CYCLE problem which is NP-complete even when input directed graphs have the maximum in-degree two and the maximum out-degree two [12]. The details are deferred to the full version.

► **Theorem 11 (*).** SHORTEST PERFECT MATCHING RECONFIGURATION is NP-hard even for bipartite graphs of maximum degree three.

The proofs actually show that SHORTEST PERFECT MATCHING RECONFIGURATION is NP-hard to approximate within a factor of less than $3/2$.



■ **Figure 5** Reduction for planar graphs of maximum degree three. Top left: a yes instance H of HAMILTONIAN CYCLE with a green Hamiltonian cycle. Top right: the constructed fragment D_v . Bottom left: The initial perfect matching M (red). Bottom middle: The target perfect matching N (blue). Bottom right: The perfect matching obtained as $M \Delta C$, where C corresponds to the Hamiltonian cycle of H .

5 Conclusion

In this paper, we studied the shortest reconfiguration problem of perfect matchings under the alternating cycle model, which is equivalent to the combinatorial shortest path problem on perfect matching polytopes. We prove that the problem can be solved in polynomial time for outerplanar graphs, but it is NP-hard, and even APX-hard for planar graphs and bipartite graphs.

Several questions remain unsolved. For polynomial-time solvability, our algorithm runs only for outerplanar graphs, and it looks difficult to extend the algorithm to other graph classes. A next step would be to try k -outerplanar graphs for fixed $k \geq 2$.

One way to tackle NP-hard cases is approximation. We only know the NP-hardness of approximating within a factor of less than $3/2$. We believe the existence of a polynomial-time constant-factor approximation. Note that we do not obtain a constant-factor approximation by flipping alternating cycles in the symmetric difference of two given perfect matchings one by one.

This paper was mainly concerned with reconfiguration of perfect matchings. Alternatively, we may consider reconfiguration of maximum matchings, or maximum-weight matchings. In those cases, we need to adopt the alternating path/cycle model. Then, the question is related to the combinatorial shortest path problem on faces of matching polytopes. Note that the perfect matching polytope is also a face of the matching polytope. Therefore, the study on maximum-weight matchings will be a generalization of this paper.

To the best of the authors' knowledge, the combinatorial shortest path problem of 0/1-polytopes has not been well investigated while the adjacency in 0/1-polytopes has been extensively studied in the literature. This paper opens up a new perspective for the study of combinatorial and computational aspects of polytopes, and connects them with the study of combinatorial reconfiguration.

References

- 1 Oswin Aichholzer, Jean Cardinal, Tony Huynh, Kolja Knauer, Torsten Mütze, Raphael Steiner, and Birgit Vogtenhuber. Flip distances between graph orientations. *CoRR*, abs/1902.06103, 2019. To appear in WG 2019. [arXiv:1902.06103](#).
- 2 Marthe Bonamy, Nicolas Bousquet, Marc Heinrich, Takehiro Ito, Yusuke Kobayashi, Arnaud Mary, Moritz Mühlenhaller, and Kunihiro Wasa. The Perfect Matching Reconfiguration Problem. *CoRR*, abs/1904.06184, 2019. To appear in MFCS 2019. [arXiv:1904.06184](#).
- 3 Nicolas Bousquet, Tatsuhiko Hatanaka, Takehiro Ito, and Moritz Mühlenhaller. Shortest Reconfiguration of Matchings. *CoRR*, abs/1812.05419, 2018. To appear in WG 2019. [arXiv:1812.05419](#).
- 4 Vašek Chvátal. On certain polytopes associated with graphs. *Journal of Combinatorial Theory, Series B*, 18(2):138–154, 1975. [doi:10.1016/0095-8956\(75\)90041-6](#).
- 5 Michael R. Garey, David S. Johnson, and Robert Endre Tarjan. The Planar Hamiltonian Circuit Problem is NP-Complete. *SIAM Journal on Computing*, 5(4):704–714, 1976. [doi:10.1137/0205049](#).
- 6 Manoj Gupta, Hitesh Kumar, and Neeldhara Misra. On the Complexity of Optimal Matching Reconfiguration. In Barbara Catania, Rastislav Kráľovic, Jerzy R. Nawrocki, and Giovanni Pighizzini, editors, *SOFSEM 2019: Theory and Practice of Computer Science — 45th International Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 27–30, 2019, Proceedings*, volume 11376 of *Lecture Notes in Computer Science*, pages 221–233. Springer, 2019. [doi:10.1007/978-3-030-10801-4_18](#).
- 7 Jan van den Heuvel. The complexity of change. In Simon R. Blackburn, Stefanie Gerke, and Mark Wildon, editors, *Surveys in Combinatorics 2013*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013. [doi:10.1017/CB09781139506748.005](#).
- 8 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12–14):1054–1065, 2011. [doi:10.1016/j.tcs.2010.12.005](#).
- 9 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012. [doi:10.1016/j.tcs.2012.03.004](#).
- 10 Denis Naddef. The Hirsch conjecture is true for $(0, 1)$ -polytopes. *Mathematical Programming*, 45(1–3):109–110, 1989. [doi:10.1007/BF01589099](#).
- 11 Naomi Nishimura. Introduction to Reconfiguration. *Algorithms*, 11(4):paper id 52, 2018. [doi:10.3390/a11040052](#).
- 12 Ján Plesník. The NP-Completeness of the Hamiltonian Cycle Problem in Planar Digraphs with Degree Bound Two. *Information Processing Letters*, 8(4):199–201, 1979. [doi:10.1016/0020-0190\(79\)90023-1](#).
- 13 Daniel Ratner and Manfred K. Warmuth. Finding a Shortest Solution for the $N \times N$ Extension of the 15-PUZZLE Is Intractable. In Tom Kehler, editor, *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, USA, August 11–15, 1986. Volume 1: Science.*, pages 168–172. Morgan Kaufmann, 1986.
- 14 Francisco Santos. A counterexample to the Hirsch Conjecture. *Annals of Mathematics*, 176(1):383–412, 2012. [doi:10.4007/annals.2012.176.1.7](#).
- 15 Noriyoshi Sukegawa. An asymptotically improved upper bound on the diameter of polyhedra. to appear in *Discrete & Computational Geometry*. [doi:10.1007/s00454-018-0016-y](#).

Closing the Gap for Pseudo-Polynomial Strip Packing

Klaus Jansen

Institut für Informatik, Christian-Albrechts-Universität zu Kiel, Germany
kj@informatik.uni-kiel.de

Malin Rau 

L’Institute Polytechnique de Grenoble, Grenoble INP, France
Malin.Rau@inria.fr

Abstract

Two-dimensional packing problems are a fundamental class of optimization problems and Strip Packing is one of the most natural and famous among them. Indeed it can be defined in just one sentence: Given a set of rectangular axis parallel items and a strip with bounded width and infinite height, the objective is to find a packing of the items into the strip minimizing the packing height. We speak of pseudo-polynomial Strip Packing if we consider algorithms with pseudo-polynomial running time with respect to the width of the strip. It is known that there is no pseudo-polynomial time algorithm for Strip Packing with a ratio better than $5/4$ unless $P = NP$. The best algorithm so far has a ratio of $4/3 + \varepsilon$. In this paper, we close the gap between inapproximability result and currently known algorithms by presenting an algorithm with approximation ratio $5/4 + \varepsilon$. The algorithm relies on a new structural result which is the main accomplishment of this paper. It states that each optimal solution can be transformed with bounded loss in the objective such that it has one of a polynomial number of different forms thus making the problem tractable by standard techniques, i.e., dynamic programming. To show the conceptual strength of the approach, we extend our result to other problems as well, e.g., Strip Packing with 90 degree rotations and Contiguous Moldable Task Scheduling, and present algorithms with approximation ratio $5/4 + \varepsilon$ for these problems as well.

2012 ACM Subject Classification Theory of computation → Packing and covering problems; Theory of computation → Scheduling algorithms

Keywords and phrases Strip Packing, pseudo-polynomial, 90 degree rotation, Contiguous Moldable Task Scheduling

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.62

Related Version A full version of the paper is available at <https://arxiv.org/abs/1712.04922>.

Funding Research was supported by German Research Foundation (DFG) project JA 612 /20-1.

1 Introduction

Two-dimensional packing problems typically have quite natural formulations and arise in a wide variety of contexts (see e.g. [8]). A characteristic challenge in this kind of problem is the space efficient placement of rectangles in a given area. Despite their simple description, they are usually quite hard and require sophisticated algorithmic techniques in order to reliably and efficiently find good solutions. Indeed, the study of algorithms for fundamental two-dimensional packing problems, like, e.g., Strip Packing, 2D-Knapsack, 2D-Bin Packing, or Unsplittable Flow on a Path, can be traced back to 1980 when Baker et al. [4] and Coffman et al [9] studied the first algorithms for two-dimensional packing problems. Furthermore, new results for these problems are regularly presented on top level conferences like FOCS, STOC and SODA up to today (see, e.g., [2, 5, 10, 13, 14, 18, 33, 27]). As all of these packing problems are NP-hard, they are typically studied in the context of approximation algorithms.



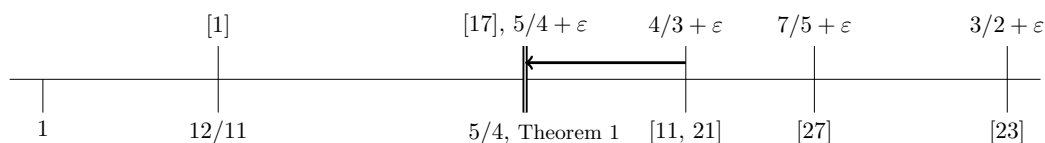
© Klaus Jansen and Malin Rau;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 62; pp. 62:1–62:14



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** The upper and lower bounds for pseudo-polynomial approximations achieved so far.

We say an approximation algorithm A has an (absolute) approximation ratio α (or call it α -approximation) if for each instance I of the problem it holds that $A(I) \leq \alpha \text{OPT}(I)$, where $\text{OPT}(I)$ is the optimal value of the corresponding objective function.

Although there is a huge range of work related to improving the absolute approximation ratio of algorithms for Strip Packing [3, 4, 6, 9, 12, 16, 22, 24, 28, 29, 30, 31] and there have been breakthroughs for 2D-Knapsack [10] and Unsplittable Flow on a Path [14], these problems are still not understood well. In the context of Strip Packing, for instance, there is a huge gap between the best known lower and upper bound of $3/2$ and $5/3 + \varepsilon$ [15], respectively. Similar, for 2D-Knapsack and Unsplittable Flow on a Path, $(1 + \varepsilon)$ -approximation schemes might be possible while the best algorithms known so far have absolute approximation ratios of $17/9 + \varepsilon$ [10] and $5/3 + \varepsilon$ [14] respectively. Closing these gaps between lower and upper bounds poses a fascinating challenge.

To close these gaps, it is essential that the corresponding problem and the structure of optimal or at least good solutions, in particular, are understood well. Hence, it can be helpful to look at the problem from different angles and consider other goals regarding the approximation or the running time. One example, where this approach has already been particularly effective, is the consideration of asymptotic approximation ratios, where we allow an extra additive term, i.e., an algorithm A has an *asymptotic* approximation ratio of α if there exists a constant c such that $A(I) \leq \alpha \text{OPT} + c$ for each instance I . While there has been extensive work on algorithms with asymptotic approximation ratios [3, 9, 12, 24, 31, 6, 22] the algorithm by Kenyon and Rémila [24] is particularly prominent. It has an asymptotic approximation ratio of $(1 + \varepsilon)\text{OPT} + \mathcal{O}(h_{\max}/\varepsilon^2)$ for each $\varepsilon > 0$ where h_{\max} is the largest occurring item height. Due to its small running time (which is a polynomial in the number of jobs as well as $1/\varepsilon$) and its relatively small additive term, the techniques used in this algorithm have become the standard to handle items which have a small height compared to the value of the objective function in most of the later developed algorithms for Strip Packing and other 2-dimensional packing problems. On the other hand, for the 2-dimensional geometric Knapsack problem, the consideration of other running times (as e.g. in [2] where the considered algorithm has a pseudo- and quasi-polynomial running time, which allows the size of the Knapsack and terms of the form $2^{\log(n)^{\mathcal{O}(1)}}$ to appear as factors in the running time) have brought new insights, which ultimately led to an algorithm for this problem (presented in [10]) that has the currently best approximation ratio of $\frac{17}{9} + \varepsilon$.

In this spirit, algorithms with pseudo-polynomial running time, which allow the widths of the strip or the size of the smallest or largest item to appear in the running time with a polynomial dependence, have been considered for the Strip Packing problem to provide a better understanding of its hardness, see Figure 1 for an overview. The so far best pseudo-polynomial time algorithm has an approximation ratio of $4/3 + \varepsilon$ [11, 21] while there is a lower bound of $5/4$ (see [17]) on the approximation ratio for this kind of algorithms unless $\text{P} = \text{NP}$.

Results

Before we summarize the results presented in this paper, we define the Strip Packing problem formally. We have to pack a set \mathcal{I} of n rectangular items into a given strip with width $W \in \mathbb{N}$ and infinite height. Each item $i \in \mathcal{I}$ has a width $w(i) \in \mathbb{N}_{\leq W}$ and a height $h(i) \in \mathbb{N}$. The area of an item $i \in \mathcal{I}$ is defined as $a(i) := h(i) \cdot w(i)$ and the area of a set of items $\mathcal{I}' \subseteq \mathcal{I}$ is defined as $a(\mathcal{I}') := \sum_{i \in \mathcal{I}'} a(i)$. A packing of the items is given by a mapping $\rho : \mathcal{I} \rightarrow \mathbb{N}_{\leq W} \times \mathbb{N}, i \mapsto (x_i, y_i)$ which assigns the lower left corner of an item $i \in \mathcal{I}$ to a position $\rho(i) = (x_i, y_i)$ in the strip. An inner point of $i \in \mathcal{I}$ (with respect to a packing ρ) is a point from the set $\text{inn}(i) := \{(x, y) \in \mathbb{R} \times \mathbb{R} \mid x_i < x < x_i + w(i), y_i < y < y_i + h(i)\}$. We say two items $i, j \in \mathcal{I}$ overlap if they share an inner point (i.e., $\text{inn}(i) \cap \text{inn}(j) \neq \emptyset$). A packing is feasible if no two items overlap and if $x_i + w(i) \leq W$ for all $i \in \mathcal{I}$. The objective of the Strip Packing problem is to find a feasible packing ρ with minimal height $h(\rho) := \max\{y_i + h(i) \mid i \in \mathcal{I}, \rho(i) = (x_i, y_i)\}$. Given an instance I of the Strip Packing problem, we denote this minimal packing height with $\text{OPT}(I)$ and dismiss the I if the instance is clear from the context.

Analyzing the structure of solutions is a valuable tool in the development of algorithms, and this holds for approximation as well as exact algorithms. By analyzing the structure of optimal solutions and finding properties that all optimal solutions share, we aim to dramatically reduce the search space of solutions in size and gain other structural insights enabling the application of well-understood algorithmic techniques like dynamic or integer programming. This general approach is widely used in the context of two-dimensional packing problems, and there are many success stories in other areas of combinatorial optimization as well. One such example is the problem of Scheduling on Identical Machines where it lead to an approximation scheme [19] whose running time (nearly) matches the lower bound [7]. In this paper, we present an analysis of the structure of optimal solutions that consist of rectangular objects placed inside a rectangular packing area, that is restricted on one side. The structural result developed from this consideration (see Lemma 3) is particularly valuable in the design of algorithms for the Strip Packing problem as we can find a pseudo-polynomial time algorithm that matches the lower bound of $5/4$ except for a negligibly small ε .

► **Theorem 1.** *There is a pseudo-polynomial algorithm for Strip Packing which finds a $(5/4 + \varepsilon)$ -approximation in $\mathcal{O}(n \log(n)) \cdot W^{\mathcal{O}_\varepsilon(1)}$ operations, where \mathcal{O}_ε dismisses all factors solely dependent on $1/\varepsilon$.*

Moreover, since we consider optimal solutions with the above described properties, this result also comes in handy for the development of algorithms for the problem Contiguous Moldable Task Scheduling, which is a generalization of Strip Packing where each rectangular item can take on a bounded number of different shapes. However, when adapting the algorithm to this problem, we get a running time where $\mathcal{O}_\varepsilon(1)$ also does appear in the exponent of the number of items n , see Theorem 2. More formally in this problem, we are given a set \mathcal{J} of n jobs and m identical machines. Each job $j \in \mathcal{J}$ can be scheduled on different numbers of machines given by $M_j \subseteq \{1, \dots, m\}$. Depending on the number of machines $i \in M_j$, each job $j \in \mathcal{J}$ has a specific processing time $p_j(i) \in \mathbb{N}$. A schedule S is given by three functions: $\sigma : \mathcal{J} \rightarrow \mathbb{N}$ which maps each job $j \in \mathcal{J}$ to a starting time $\sigma(j)$; $\rho : \mathcal{J} \rightarrow \{1, \dots, m\}$ which maps each job $j \in \mathcal{J}$ to the number of processors $\rho(j) \in M_j$ it is processed on; and $\varphi : \mathcal{J} \rightarrow \{1, \dots, m\}$ which maps each job $j \in \mathcal{J}$ to the first machine it is processed on. The job $j \in \mathcal{J}$ will use the machines $\varphi(j)$ to $\varphi(j) + \rho(j) - 1$ contiguously. A schedule $S = (\sigma, \rho, \varphi)$ is feasible if each machine processes at most one job at a time and its makespan is defined by $\max_{j \in \mathcal{J}} \sigma(j) + p_j(\rho(j))$. The objective is to find a feasible schedule, which minimizes the makespan. This problem and prominent variants where the jobs do not need to occupy contiguous machines have been widely studied, see e.g. [32, 25, 26, 23, 20].

62:4 Closing the Gap for Pseudo-Polynomial Strip Packing

This problem is a generalization of Strip Packing as it contains this problem (and Strip Packing with rotations) as a special case: We define the number of machines m as the width of the strip W and for each item $i \in \mathcal{I}$ we introduce one job i with $M_i := \{w(i)\}$ and processing time $p_i(w(i)) = h(i)$ (or introduce one job i with $M_i := \{w(i), h(i)\}$ and processing times $p_i(w(i)) = h(i)$ and $p_i(h(i)) = w(i)$ respectively). Therefore, we cannot hope for a pseudo-polynomial algorithm with a ratio better than $5/4$ unless $P = NP$. We adapt the structure and algorithmic result to find an algorithm with an approximation ratio, which almost matches this bound.

► **Theorem 2.** *There is a pseudo-polynomial algorithm for the Contiguous Moldable Parallel Tasks Scheduling problem which finds a $(5/4 + \varepsilon)$ -approximation in $(nm)^{\mathcal{O}_\varepsilon(1)}$ operations.*

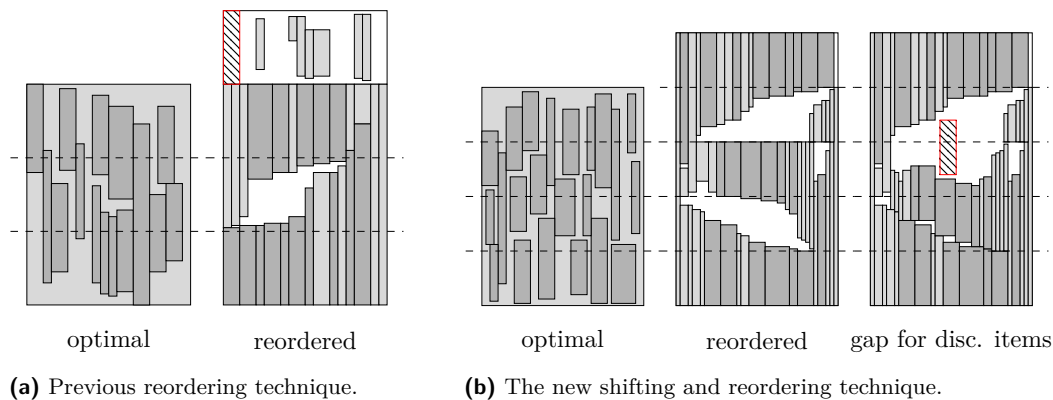
Remark that for the case where for at least one job $j \in \mathcal{J}$ we have that $|M_j| \in \Omega(m)$ the running time of this algorithm is polynomial in the input size. Furthermore, we can hope that in realistic instances the number of machines is bounded by a function in the number of jobs n . If this is the case, the mentioned algorithm is a polynomial time algorithm as well, which further motivates the consideration of pseudo-polynomial time algorithms. As the Contiguous Moldable Parallel Tasks Scheduling contains the Strip Packing with Rotations as a special case, this theorem implies a $(5/4 + \varepsilon)$ -approximation with running time $(nW)^{\mathcal{O}_\varepsilon(1)}$ for this problem as well.

Methodology

We follow the general approach mentioned above. More precisely, we analyze optimal solutions and how they can be transformed carefully into well-structured solutions without too much loss in the objective. Knowing that such a transformation is always attainable, the algorithm will iterate the potential structures of the transformed optimal packings and fill the items inside this structure using dynamic and linear programming. The same basic scheme has been used for this and other packing problems before, e.g. [2, 27, 11, 21]. However, finding a suitable transformation to a well structured solution provides a challenge that depends on the problem itself (i.e. a structural result from other packing problems might not be applicable for Strip Packing) and our approach significantly differs from previous ones.

In the approaches seen before, i.e., in [27], [11] and [21], there arises a natural set of critical items, e.g., all items with height larger than $1/3 \text{ OPT}$ in [11] and [21]. The characteristic of this set is that the aspired approximation ratio is exceeded if we place one of these items on top of the optimal packing area. The transformation strategy used in these previous approaches is heavily dependent on the fact that there can be at most two critical items on top of each other. This allows placing all critical items in the optimal packing area while discarding some noncritical items, which are placed on top of the optimal packing later (see Figure 2a). If three critical items can be put on top of each other (which will be the case as soon as a critical item can have a height smaller than $\text{OPT}/3$) the described transformation will not work. To find an algorithm with ratio $4/3 - \varepsilon$, we need to overcome this major obstacle.

To construct a $(5/4 + \varepsilon)$ -approximation, we introduce a new technique, in the following called **shifting and reordering**. In contrast to the previous results, our structural result does not guarantee that all critical items are packed inside the optimal packing area. Instead, we shift and reorder the items of an optimal packing such that the critical items with height larger than $1/4 \text{ OPT}$ are aligned into three shelves using the area $W \times (5/4 + \varepsilon) \text{ OPT}$ (see Figure 2b).



■ **Figure 2** A comparison of old and new strategies in the simplified case. Dark gray rectangles represent the critical items, while the light gray area represents the other items, which can be sliced vertically during the reordering. The hatched area represents an area where we can place the items that are sliced by the reordering.

A challenge which arises using this new strategy is the fact that by the newly introduced shifting and reordering technique a constant number of the other (non-critical) items will be sliced vertically and thus have to be discarded temporarily from the packing. Although this set of discarded items also appears in previous approaches, their handling differs significantly. Since the shifting strategy extends the occupied packing area by the factor $(5/4 + \varepsilon)$ with respect to its height, these discarded items cannot be placed on top of the packing as done in previous approaches, see Figure 2. Instead, the discarded items have to be placed carefully into gaps generated by the shifting and reordering step. By a careful analysis of the rearrangement, we prove that each possible structure of a rearranged optimal packing provides suitable gaps to place these items.

In Section 3, we present the central idea to find the improved structural result – the shifting and reordering technique. However, to highlight the basic steps, a simplified problem is considered. In this simplified case just the critical items have to be placed integrally while all other items are allowed to be partitioned into vertical slices, which do not have to be placed contiguously.

In general (when the non critical items cannot be placed as non contiguous vertical slices) the slicing of some items will cause problems when trying to place them inside gaps generated by the new strategy, because these gaps might be thin. Hence, we cannot slice items that are too wide in some sense. Nevertheless, we may slice certain narrow items further called sliceable. To overcome this obstacle, we use a lemma from [21], which states the possibility to partition each slightly stretched packing into $\mathcal{O}_\varepsilon(1)$ rectangular areas (without removing any item). This partition provides the property that each critical item is contained in (or intersected by) area(s) exclusively containing critical and sliceable items. Up to three critical items can overlap each of the vertical borders of these areas and these overlapping items may not be shifted horizontally or vertically by our new technique. In the full version, we extend the strategy presented in Section 3 to these areas although it becomes much more involved.

Combining our new techniques to place critical items on three shelves, find suitable gaps for discarded non-critical items and handle the exclusive slicing of narrow items together enables us to prove the structural result from Lemma 3 and in Section 2, we provide a more detailed road-map of its proof. As mentioned above, the algorithm iterates all possible structures defined by the structure result and tries to place all items into this structure using linear and dynamic programming until a suitable structure is found.

The structural result applies to all optimal solutions with the property that they consist of rectangular objects placed into a rectangle that is extendable on one side. Optimal solutions of the three considered problems, i.e., Strip Packing, Strip Packing with rotations and Contiguous Moldable Task Scheduling, all have this property. Thanks to this fact, we were able to obtain algorithms which find $5/4 + \varepsilon$ approximations for each of the three problems by carefully adapting the underlying dynamic program.

2 Structural Result

In this section, we introduce the *Structural Lemma*, which presents the fundamental insight to achieve the approximation ratio $(5/4 + \varepsilon)$. Roughly speaking, the lemma states that each optimal solution can be transformed such that it has a simple structure, see Lemma 3. Due to space limitations, we cannot present the proof here and we refer to the full version. Nevertheless, we provide a high-level overview on the steps of the proof, which consists of the following two basic steps. First the given instance and a corresponding optimal solution is simplified by rounding the sizes of the items (widths and heights) as well as partitioning the set of items into parts, that can be handled almost independently. Afterward, the items in the optimal packing are reordered such that they provide the properties demanded by the lemma.

Given an optimal packing with height OPT for an instance I , we first perform some simplification steps. First, we partition the set of items by defining

- $\mathcal{L} := \{i \in \mathcal{I} \mid h(i) > \delta\text{OPT}, w(i) \geq \delta W\}$ as the set of large items,
- $\mathcal{T} := \{i \in \mathcal{I} \mid h(i) \geq (1/4 + \varepsilon)\text{OPT}, w(i) < \delta W\}$ as the set of tall items,
- $\mathcal{V} := \{i \in \mathcal{I} \mid \delta\text{OPT} \leq h(i) < (1/4 + \varepsilon)\text{OPT}, w(i) \leq \mu W\}$ as the set of vertical items,
- $\mathcal{M}_V := \{i \in \mathcal{I} \mid \varepsilon\text{OPT} \leq h(i) < (1/4 + \varepsilon)\text{OPT}, \mu W < w(i) \leq \delta W\}$ as the set of vertical medium items,
- $\mathcal{H} := \{i \in \mathcal{I} \mid h(i) \leq \mu\text{OPT}, \delta W \leq w(i)\}$ as the set of horizontal items,
- $\mathcal{S} := \{i \in \mathcal{I} \mid h(i) \leq \mu\text{OPT}, w(i) \leq \mu W\}$ as the set of small items and
- $\mathcal{M} := \{i \in \mathcal{I} \mid h(i) < \varepsilon\text{OPT}, \mu W < w(i) \leq \delta W\} \cup \{i \in \mathcal{I} \mid \mu\text{OPT} < h(i) \leq \delta\text{OPT}\} = \mathcal{I} \setminus (\mathcal{L} \cup \mathcal{T} \cup \mathcal{V} \cup \mathcal{M}_V \cup \mathcal{H} \cup \mathcal{S})$ as the set of medium sized items,

where we chose δ and μ such that the total area of the items $\mathcal{M}_V \cup \mathcal{M}$ is small, resulting in $|\mathcal{M}_V|$ to be in $\mathcal{O}(1/(\varepsilon\delta^2))$. Afterward the heights of the items with height larger than δOPT are rounded such that there are at most $\mathcal{O}(1/(\varepsilon\delta))$ sizes and such that their y -coordinates are positioned on multiples of $\varepsilon\delta\text{OPT}$.

In the next step, we discard the items $\mathcal{S} \cup \mathcal{M}$ from the packing since they can be placed later on, using the NFDH algorithm from [9]. By an adaption of a lemma in [21], we were able to show that the residual packing can be partitioned into $\mathcal{O}_\varepsilon(1)$ rectangular subareas, called boxes, that contain exactly one item from the set $\mathcal{L} \cup \mathcal{M}_V$, only items from the set \mathcal{H} , or only items from the set $\mathcal{T} \cup \mathcal{V}$. Furthermore, horizontal items are allowed to overlap horizontal box borders, while vertical and tall items are allowed to overlap vertical box borders. Note that in this partitioning step no item is removed from the packing or changes its position.

Remark that in [21] the items in \mathcal{M}_V were handled the same as the medium sized items \mathcal{M} , i.e., they were simply placed on the top of the packing. However, this is not possible in our case since these items can have a height of up to $(1/4 + \varepsilon)\text{OPT}$ and we need the extra height of $(1/4 + \varepsilon)\text{OPT}$ to apply the shifting and reordering. Consequently, we have to think of a new strategy to handle them. Since their number is bounded by $\mathcal{O}_\varepsilon(1)$, it is possible to handle them as if they were large. This different handling of vertical medium items \mathcal{M}_V is, regarding previous algorithms, one of the novelties of this result.

Next, we consider the mentioned partition of the optimal solution into rectangular axis-parallel boxes. The items in $\mathcal{L} \cup \mathcal{M}_V$ and the boxes containing horizontal items need no more attention since for each item in $\mathcal{L} \cup \mathcal{M}_V$ we can guess its position in pseudo-polynomial time and by extending the packing by a factor of $\mathcal{O}(\varepsilon)$ the horizontal items can be placed inside the boxes using a configuration LP building upon the techniques presented in [24].

We innovate the reordering of the items inside the boxes for vertical and tall items $\mathcal{T} \cup \mathcal{V}$ using the new *shifting and reordering* technique (see Section 3). Using this technique, we extend all the boxes with height larger than $\text{OPT}/2$ by only $\text{OPT}/4$, shift and reorder the items inside, and partition their area such that each subarea contains either only tall items of the same height, only vertical items, or no item. Note that the boxes can be overlapped by up to three tall items on each side (left or right). When reordering the items inside the boxes, we cannot move these overlapping items. We refer to the full version for the proof of this alteration with overlapping items.

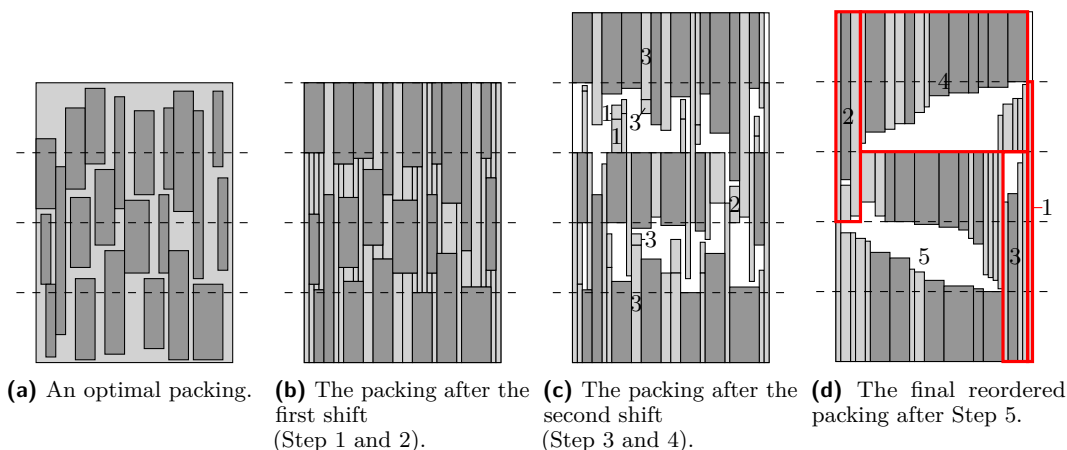
During this reordering step, we slice vertical items vertically. This slicing needs to be fixed since in the aspired Structural Lemma 3 all the items are positioned integral. We prove that by using a configuration LP to place the vertical items, we end up with only $\mathcal{O}_\varepsilon(1)$ items, that have to be placed fractionally. We place these items inside $\mathcal{O}_\varepsilon(1)$ containers of width μW and height $\text{OPT}/4$. An arising challenge is the placement of these containers inside the already extended packing. Other than in the previous attempts (see [27], [11], or [21]), it is not possible to place these extra boxes on top of the packing. By a careful analysis of the area added due to the shifting step, we manage to find a placement of these items inside the rearranged packing. All these considerations together are enough to prove the following structural result:

► **Lemma 3 (Structural Lemma).** *By extending the packing height to $(5/4 + 5\varepsilon)\text{OPT}$ each rounded optimal packing can be rearranged and partitioned into $\mathcal{O}(1/(\delta^3\varepsilon^5))$ boxes with the following properties:*

- *There are $|\mathcal{L}| + |\mathcal{M}_V| = \mathcal{O}(1/(\delta^2\varepsilon))$ boxes $\mathcal{B}_\mathcal{L}$ each containing exactly one item from the set $\mathcal{L} \cup \mathcal{M}_V$ and all items from this set are contained in these boxes.*
- *There are at most $\mathcal{O}(1/(\delta^2\varepsilon))$ boxes $\mathcal{B}_\mathcal{H}$ containing all horizontal items \mathcal{H} with $\mathcal{B}_\mathcal{H} \cap \mathcal{B}_\mathcal{L} = \emptyset$. The horizontal items can overlap horizontal box borders, but never vertical box borders.*
- *There are at most $\mathcal{O}(1/(\delta^2\varepsilon^5))$ boxes $\mathcal{B}_\mathcal{T}$ containing tall items, such that each tall item t is contained in a box with rounded height $h(t)$.*
- *There are at most $\mathcal{O}(1/(\delta^3\varepsilon^5))$ boxes $\mathcal{B}_\mathcal{V}$ containing vertical items, such that each vertical item v is contained in a box with rounded height $h(v)$.*
- *There are at most $\mathcal{O}(1/(\delta^2\varepsilon^5))$ boxes $\mathcal{B}_\mathcal{S}$ for small items, such that the total area of these boxes combined with the total free area inside the horizontal boxes is at least as large as the total area of the small items.*
- *The lower and top border of each box is positioned at a multiple of $\varepsilon\delta\text{OPT}$.*

3 Introducing the Shifting and Reordering Technique

To demonstrate the central new idea which leads to the improved structural result – the shifting and reordering technique – we consider the following simplified case. We have to pack items with a tall height integrally, while we are allowed to slice all other items vertically. We can assume that the packing, which we consider here, is the packing inside a box for \mathcal{T} and \mathcal{V} for the case that no tall item overlaps the box borders. Remember, in the general case, there can be such items and hence the reordering gets a little bit more complicated as in this simplified case. We will demonstrate that, in this simplified scenario, it is possible to rearrange the items such that there are a constant number of rectangular subareas, which contain only tall items with the same height.



■ **Figure 3** States of the item rearrangement. Dark rectangles represent tall items while light gray areas might contain non-tall sliced items.

Let a packing with height H be given. We define tall items as the items which have a height larger than $\frac{1}{4}H$. Further, assume that there is an arithmetic grid with $N + 1$ horizontal grid lines with distance H/N such that each tall item starts and ends at the grid lines. For now, we can think of this grid as the integral grid with $H + 1$ grid lines. Later, we can reduce the grid lines by rounding the heights of the items. We are interested in a fractional packing of the non-tall items. Therefore, we replace each non-tall item i by exactly $w(i)$ items with height $h(i)$ and width 1. This step is called slicing. We define a box as a rectangular subarea of the packing area.

► **Lemma 4.** *By adding at most $\frac{1}{4}H$ to the packing height and slicing non-tall items, we can rearrange the items such that we generate at most $\frac{3}{2}N$ containers which contain tall items with the same height only, and at most $\frac{9}{4}N + 1$ container for sliced items.*

Proof. In this proof, we will present a rearrangement strategy which provides the desired properties. This strategy consists of two shifting steps and one reordering step. In the shifting steps, we shift items in the vertical direction, while in the reordering step we change the item positions horizontally. In the first shifting step, we ensure that tall items intersecting the horizontal lines $\frac{1}{4}H$ or $\frac{3}{4}H$ will touch the bottom or the top of the packing area, respectively. In the second shift, we ensure that tall items not intersecting these lines have a common upper border as well. Last, we reorder the items such that tall items with the same height are positioned next to each other if they have a common upper or lower border.

Step 1: First Shift. Note that there is no tall item completely below $\frac{1}{4}H$ or completely above $\frac{3}{4}H$ since each tall item has a height larger than $\frac{1}{4}H$. We shift each tall item t intersecting the horizontal line $\frac{1}{4}H$ down, such that its bottom border touches the bottom of the strip. The sliced items below t are shifted up exactly $h(t)$, such that they are now positioned above t . In the same way, we shift each tall item intersecting the horizontal line at $\frac{3}{4}H$ but not the horizontal line at $\frac{1}{4}H$ such that its upper border is positioned at H and shift the sliced items down accordingly, see Figure 3b.

Step 2: Introducing Pseudo Items. At this point, we introduce a set of containers for the sliced items, which we call pseudo items, see Figure 3b. We draw vertical lines at each left or right border of a tall item and erase these lines on any tall item. Each area between two

consecutive lines which is bounded on top and bottom by a tall item or the packing area and contains sliced items represents a new item called pseudo item. Note that no sliced item is intersecting any box border since they are positioned on integral widths only. Furthermore, when we shift a pseudo item, we shift all sliced items included in this container as well.

When constructing the pseudo items, we consider one special case. Consider a tall item t with height larger than $\frac{3}{4}H$. There can be no tall item positioned above or below t , and t was shifted down. For these items, we introduce one pseudo item of height H and width $w(t)$ containing t and all sliced items above. Note that each pseudo item has a height, which is a multiple of H/N . Furthermore, note that each tall or pseudo item touching the top or the bottom border of the packing area has a height larger than $\frac{1}{4}H$.

Step 3: Second Shift. Next, we do a second shifting step consisting of three sub-steps. First, we shift each tall or pseudo item intersected by the horizontal line at $\frac{3}{4}H$ but not the horizontal line at $\frac{1}{4}H$ exactly $\frac{1}{4}H$ upwards. Second, we shift each pseudo item positioned between the horizontal lines at $\frac{1}{2}H$ and $\frac{3}{4}H$, such that their lower border is positioned at the horizontal line $\frac{3}{4}H$. Last, we shift each tall or pseudo item intersected by the horizontal line at $\frac{1}{2}H$ but not the horizontal line at $\frac{1}{4}H$ or $\frac{3}{4}H$ such that its upper border is positioned at the horizontal line $\frac{3}{4}H$. After this shifting, no item overlaps another item since we have shifted the items intersecting the line at $\frac{3}{4}H$ exactly $\frac{1}{4}H$, while each item below is shifted at most $\frac{1}{4}H$.

Step 4: Fusing Pseudo Items. After the second shift, we will fuse and shift some pseudo items. We want to establish the property that each tall and pseudo item has one border (upper or lower), which touches one of the horizontal lines at 0 , $\frac{3}{4}H$, or $\frac{5}{4}H$. At the moment there can be some pseudo items between the horizontal lines $\frac{1}{4}H$ and $\frac{1}{2}H$, which do not touch one of the three lines. In the following, we study the three cases where those pseudo items can occur. These items do only exist if there is a tall item touching the bottom of the packing and another tall item above this item with a lower border at or below $\frac{1}{2}H$ before the second shifting step. Consider two consecutive vertical lines we had drawn to generate the pseudo items. If a tall item overlaps the vertical strip between these lines, its right and left borders either lie on the strips borders or outside of the strip.

Case 1: In the first considered case there are three tall items, t_1 , t_2 , and t_3 from bottom to top, which overlap the strip. In this scenario t_1 must have its lower border at 0 , t_2 its upper border at $\frac{3}{4}H$, and t_3 its upper border at $\frac{5}{4}H$. As a consequence, there are at most two pseudo items: One is positioned between t_1 and t_2 , and the other between t_2 and t_3 . We will stack them, such that the lower border of the stack is positioned at $\frac{3}{4}H$ and prove that this is possible without overlapping t_3 . The total height of both pseudo items is $H - h(t_1) - h(t_2) - h(t_3)$. The total area not occupied by tall items is $H - h(t_1) - h(t_2) - h(t_3) + \frac{1}{4}H$ since we have added $\frac{1}{4}H$ to the packing height. The distance between t_1 and t_2 is at most $\frac{1}{4}H$ since t_1 's lower border is at 0 and t_2 's upper border is at $\frac{3}{4}H$ and both have a height larger than $\frac{1}{4}H$. Therefore, the distance between t_2 and t_3 is at least $H - h(t_1) - h(t_2) - h(t_3)$, see Figure 3c at the items marked with 1.

Case 2: Now consider the case where there is one tall item t_1 touching the bottom, and one tall item t_2 with height at least $\frac{1}{2}H$ touching $\frac{5}{4}H$. Obviously, t_2 has a height of at most $\frac{3}{4}H$. Furthermore, there is at most one pseudo item, and it has to be positioned between $\frac{1}{4}H$ and $\frac{1}{2}H$. We shift this pseudo item up until its bottom touches $\frac{1}{2}H$, see Figure 3c

62:10 Closing the Gap for Pseudo-Polynomial Strip Packing

at the item marked with 2. This is possible without constructing any overlap, because the distance between t_1 and the horizontal line $1/2H$ is less than $1/4H$ and, therefore, the distance between the line $1/2H$ and the lower border of the tall item is larger than the height of the pseudo item.

After this step, we consider each tall item t with height larger than $1/2H$ touching $5/4H$. We generate a new pseudo item with width $w(t)$ and height $3/4H$, with upper border at $5/4H$ and lower border at $1/2H$, containing all pseudo items below t touching $1/2H$ with their lower border.

Case 3: In the last case we consider, there are two tall items t_1 and t_2 and two pseudo items; one of the items t_1 and t_2 touches the top of the packing or the bottom, while the other ends at $3/4H$. Hence, the distance between the tall items has to be smaller than $1/4H$. Furthermore, one of the pseudo items has to touch the top or the bottom of the packing while the other is positioned between t_1 and t_2 . Since the distance between t_1 and t_2 is less than $1/4H$, one of the distances between the packing border and the lower border of t_1 or the upper border of t_2 is at least $H - h(t_1) - h(t_2)$. Therefore, we can fuse both pseudo items by shifting the one between t_1 and t_2 such that it is positioned above or below the other one, see Figure 3c at the items marked with 3.

► **Observation 1.** *After the shifting and fusing, each tall or pseudo item touches one of the horizontal lines at 0, $3/4H$ or $5/4H$.*

Step 5: Reordering the Items. In the last part of the rearrangement, we reorder the items horizontally to place pseudo and tall items with the same height next to each other. In this reordering step, we create five areas each reserved for certain items. To do so, we take vertical slices of the packing and move them to the left or the right in the strip. A vertical slice is an area of the packing with width one and height of the considered packing area, i.e. $5/4H$ in this case. While rearranging these slices, it will never happen that two items overlap. However, it can happen, that some of the tall items are placed fractionally afterward. This will be fixed in later steps.

Area 1: First, we will extract all vertical slices containing (pseudo) items with height H . Then, shifting all the remaining vertical slices to the left as much as possible, we create one box for pseudo items of height H at the right, see Figure 3d at Area 1. In this area, we sort the pseudo items such that the pseudo items containing tall items with the same height are placed next to each other. In this step, we did not place any tall item fractionally.

Area 2: Afterward, we take each vertical slice containing a (pseudo) item with height at least $1/2H$ touching the horizontal line at $5/4H$. Remember, there might be pseudo items containing a tall item t with a height between $1/2H$ and $3/4H$. We shift these slices to the left of the packing and sort them in descending order of the tall items height $h(t)$, see Figure 3d at Area 2. Afterward, we sort the pseudo items below these tall items, which are touching $1/2H$ with their bottom in ascending order of their heights, which is possible without generating any overlapping. In this step, it can happen that we slice tall items which touch the bottom of the strip. We will fix this slicing in one of the following steps, when we consider Area 5.

Area 3: Next, we look at vertical slices containing (pseudo) items t with height at least $1/2H$ touching the bottom of the strip. We shift them to the right until they touch the Area 1 and sort these slices in ascending order of the heights $h(t)$, see Figure 3d at Area 3.

Note that there are no pseudo or tall items that have their upper border positioned at $\frac{3}{4}H$ in these slices. In this step, it can happen that we slice tall items touching the top of the packing. This will be fixed in the next step.

Area 4: Look at the area above $\frac{3}{4}H$ and left of Area 2 but right of Area 1, see Figure 3d at Area 4. In this area no item overlaps the horizontal line $\frac{3}{4}H$. Therefore, we have a rectangular area where each item either touches its bottom or its top and no item is intersected by the area's borders. In [27] it was shown that, in this case, we can sort the items touching the line $\frac{3}{4}H$ in ascending order of their height and the items touching $\frac{5}{4}H$ in descending order of heights and no item will overlap another item. Now all items with the same height are placed next to each other, and thus we fixed the slicing of tall items.

Area 5: In the last step, we will reorder the remaining items. Namely the items touching the bottom of the strip left of Area 3 and the items touching the horizontal line at $\frac{3}{4}H$ with their top between Area 2 and Area 3. The items touching the bottom are sorted in descending order of their height and the items touching the horizontal line at $\frac{3}{4}H$ are sorted in ascending order regarding their heights.

▷ **Claim.** After the reordering of Area 5 no item overlaps another.

Proof. First, note that the items touching $\frac{5}{4}H$ have a height of at most $\frac{3}{4}H$. Therefore, no item touching the bottom having height at most $\frac{1}{2}H$ can overlap with these items. Furthermore, note that before the reordering no item was overlapping another. Let us assume there are two items b and t , which overlap at a point (x, y) after this reordering. Then all items left of x touching $\frac{3}{4}H$ have their lower border below y , while all items touching the bottom left of x have their upper border above y . Therefore, at every point left and right of (x, y) in the Area 5 there is an item overlapping it. Hence, the total width of items overlapping the horizontal line y is larger than the width of the Area 5. Therefore in the original ordering, there would have been items overlapping each other already since we did not add any items – a contradiction. As a consequence in this new ordering, no two items overlap, which concludes the proof of the claim. ◁

Analyzing the Number of Constructed Boxes. In the last part of this proof, we analyze how many boxes we have created for tall and sliced items.

▷ **Claim.** After the described reordering there are at most $\frac{3}{2}N$ boxes for tall items all containing items of only one height.

Proof. Each tall item with height at least $\frac{3}{4}H$ touches the bottom and we create at most one box in Area 1 for each height. Therefore, we create at most $N/4$ boxes for these items. Each tall item that has a height between $\frac{1}{2}H$ and $\frac{3}{4}H$ touches either the bottom or the horizontal line $\frac{5}{4}H$. On each of these lines, we create at most one box for items with the same height. Therefore, we create at most $2N/4$ boxes for these items. Last, each tall item with height larger than $\frac{1}{4}H$ but smaller than $\frac{1}{2}H$ either touches the bottom of the packing, the horizontal line $\frac{3}{4}H$ or the horizontal line $\frac{5}{4}H$. At each of these lines, we create at most one box for each height. Therefore, we create at most $3N/4$ of these boxes. In total, we create at most $\frac{3}{2}N$ boxes for tall items. ◁

▷ **Claim 5.** After the described reordering there are at most $\frac{9}{4}N + 1$ boxes containing sliced items.

62:12 Closing the Gap for Pseudo-Polynomial Strip Packing

Proof. Let us consider the number of boxes for sliced items. Each pseudo item's height is a multiple of H/N . Therefore, we have at most N different sizes for pseudo items. There are at most 4 boxes for each height less than $1/4H$. One is touching H with its top border in Area 1, one is touching $3/4H$ with its bottom border in Area 4, one is touching $3/4H$ with its top border in Area 5, and one is touching $1/2H$ with its bottom border in Area 2. Furthermore, there are at most 3 boxes for each size between $1/4H$ and $1/2H$. One is touching $5/4H$ with its top border in Area 4, one is touching $3/4H$ with its top border in Area 5, and one is touching 0 with its bottom border in Area 5. Additionally, there are at most 2 boxes for each pseudo item size larger than $1/2H$. One is touching $5/4H$ with its top border in Area 2, the other is touching 0 with its bottom border in Area 3. Last there is only one pseudo item with height larger than $3/4H$ in Area 1. It has height H . Since the grid is arithmetically defined, we have at most $N/4$ sizes with height at most $1/4H$, $N/4$ sizes between $1/4H$ and $1/2H$ and at most $1/4N$ sizes between $1/2H$ and $3/4H$. Therefore, we create at most $4 \cdot 1/4N + 3 \cdot 1/4N + 2 \cdot 1/4N + 1 = \frac{9}{4}N + 1$ boxes for sliced items. \triangleleft

Since the number of boxes for tall and sliced items is as small as claimed, this concludes the proof of the Lemma 4. \blacktriangleleft

In this section, we have proven that in this simplified case it is possible to reorder the items such that they have a nice structure. Nevertheless, when considering the mentioned partition into rectangular subareas (called boxes) from [21] we encounter some obstacles. In each box B containing tall and vertical items there can be up to three tall items overlapping its left and right border. Especially critical to apply the above described shifting and reordering technique are the items overlapping the box at the height $h(B) - H/4$, because these items cannot be moved. The reason why we cannot move these objects is the impossibility of judging their intertwining with other objects within the respective other box(es) in which they are contained. Therefore, since the reordering technique requires them to be shifted up, a first step is to discard these items from the respective boxes by partitioning them into smaller boxes at the left and right borders of these items. However, we cannot get rid of the items overlapping the box below this horizontal line of $h(B) - H/4$, and handling these items without moving them becomes quite technical. A detailed analysis how to reorder the items inside a box if tall items overlap the borders can be found in the full version of this paper.

References

- 1 Anna Adamaszek, Tomasz Kociumaka, Marcin Pilipczuk, and Michal Pilipczuk. Hardness of Approximation for Strip Packing. *TOCT*, 9(3):14:1–14:7, 2017. doi:10.1145/3092026.
- 2 Anna Adamaszek and Andreas Wiese. A quasi-PTAS for the Two-Dimensional Geometric Knapsack Problem. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1491–1505, 2015. doi:10.1137/1.9781611973730.98.
- 3 Brenda S. Baker, Donna J. Brown, and Howard P. Katseff. A $5/4$ Algorithm for Two-Dimensional Packing. *Journal of Algorithms*, 2(4):348–368, 1981. doi:10.1016/0196-6774(81)90034-1.
- 4 Brenda S. Baker, Edward G. Coffman Jr., and Ronald L. Rivest. Orthogonal Packings in Two Dimensions. *SIAM Journal on Computing*, 9(4):846–855, 1980. doi:10.1137/0209064.
- 5 Nikhil Bansal and Arindam Khan. Improved Approximation Algorithm for Two-Dimensional Bin Packing. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 13–25, 2014. doi:10.1137/1.9781611973402.2.

- 6 Marin Bougeret, Pierre-François Dutot, Klaus Jansen, Christina Robenek, and Denis Trystram. Approximation Algorithms for Multiple Strip Packing and Scheduling Parallel Jobs in Platforms. *Discrete Mathematics, Algorithms and Applications*, 3(4):553–586, 2011. doi:10.1142/S1793830911001413.
- 7 Lin Chen, Klaus Jansen, and Guochuan Zhang. On the optimality of approximation schemes for the classical scheduling problem. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 657–668, 2014. doi:10.1137/1.9781611973402.50.
- 8 Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 2017. doi:10.1016/j.cosrev.2016.12.001.
- 9 Edward G. Coffman Jr., Michael R. Garey, David S. Johnson, and Robert Endre Tarjan. Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980. doi:10.1137/0209062.
- 10 Waldo Gálvez, Fabrizio Grandoni, Sandy Heydrich, Salvatore Ingala, Arindam Khan, and Andreas Wiese. Approximating Geometric Knapsack via L-Packings. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 260–271, 2017. doi:10.1109/FOCS.2017.32.
- 11 Waldo Gálvez, Fabrizio Grandoni, Salvatore Ingala, and Arindam Khan. Improved Pseudo-Polynomial-Time Approximation for Strip Packing. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 9:1–9:14, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.9.
- 12 Igal Golan. Performance Bounds for Orthogonal Oriented Two-Dimensional Packing Algorithms. *SIAM Journal on Computing*, 10(3):571–582, 1981. doi:10.1137/0210042.
- 13 Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. To Augment or Not to Augment: Solving Unsplittable Flow on a Path by Creating Slack. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2411–2422, 2017. doi:10.1137/1.9781611974782.159.
- 14 Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. A $(5/3 + \epsilon)$ -approximation for unsplittable flow on a path: placing small tasks into boxes. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 607–619, 2018. doi:10.1145/3188745.3188894.
- 15 Rolf Harren, Klaus Jansen, Lars Prädél, and Rob van Stee. A $(5/3 + \epsilon)$ -approximation for strip packing. *Computational Geometry*, 47(2):248–267, 2014. doi:10.1016/j.comgeo.2013.08.008.
- 16 Rolf Harren and Rob van Stee. Improved Absolute Approximation Ratios for Two-Dimensional Packing Problems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 5687 of *Lecture Notes in Computer Science*, pages 177–189. Springer, 2009. doi:10.1007/978-3-642-03685-9_14.
- 17 Sören Henning, Klaus Jansen, Malin Rau, and Lars Schmarje. Complexity and Inapproximability Results for Parallel Task Scheduling and Strip Packing. In *Computer Science - Theory and Applications - 13th International Computer Science Symposium in Russia, CSR 2018, Moscow, Russia, June 6-10, 2018, Proceedings*, pages 169–180, 2018. doi:10.1007/978-3-319-90530-3_15.
- 18 Sandy Heydrich and Andreas Wiese. Faster approximation schemes for the two-dimensional knapsack problem. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 79–98, 2017. doi:10.1137/1.9781611974782.6.
- 19 Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the Gap for Makespan Scheduling via Sparsification Techniques. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 72:1–72:13, 2016. doi:10.4230/LIPIcs.ICALP.2016.72.

- 20 Klaus Jansen and Felix Land. Scheduling Monotone Moldable Jobs in Linear Time. In *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, Vancouver, BC, Canada, May 21-25, 2018*, pages 172–181, 2018. doi:10.1109/IPDPS.2018.00027.
- 21 Klaus Jansen and Malin Rau. Improved approximation for two dimensional strip packing with polynomial bounded width. In *WALCOM: Algorithms and Computation*, volume 10167 of LNCS, pages 409–420, 2017. doi:10.1007/978-3-319-53925-6_32.
- 22 Klaus Jansen and Roberto Solis-Oba. Rectangle packing with one-dimensional resource augmentation. *Discrete Optimization*, 6(3):310–323, 2009. doi:10.1016/j.disopt.2009.04.001.
- 23 Klaus Jansen and Ralf Thöle. Approximation Algorithms for Scheduling Parallel Jobs. *SIAM Journal on Computing*, 39(8):3571–3615, 2010. doi:10.1137/080736491.
- 24 Claire Kenyon and Eric Rémila. A Near-Optimal Solution to a Two-Dimensional Cutting Stock Problem. *Mathematics of Operations Research*, 25(4):645–656, 2000. doi:10.1287/moor.25.4.645.12118.
- 25 Walter Ludwig and Prasoon Tiwari. Scheduling Malleable and Nonmalleable Parallel Tasks. In *5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 167–176, 1994.
- 26 Gregory Mounie, Christophe Rapine, and Denis Trystram. A $3/2$ -Approximation Algorithm for Scheduling Independent Monotonic Malleable Tasks. *SIAM J. Comput.*, 37(2):401–412, 2007. doi:10.1137/S0097539701385995.
- 27 Giorgi Nadiradze and Andreas Wiese. On approximating strip packing with a better ratio than $3/2$. In *27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1491–1510, 2016. doi:10.1137/1.9781611974331.ch102.
- 28 Ingo Schiermeyer. Reverse-Fit: A 2-Optimal Algorithm for Packing Rectangles. In *2nd Annual European Symposium on Algorithms (ESA) - Algorithms*, pages 290–299, 1994. doi:10.1007/BFb0049416.
- 29 Daniel Dominic Sleator. A 2.5 Times Optimal Algorithm for Packing in Two Dimensions. *Information Processing Letters*, 10(1):37–40, 1980. doi:10.1016/0020-0190(80)90121-0.
- 30 A. Steinberg. A Strip-Packing Algorithm with Absolute Performance Bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997. doi:10.1137/S0097539793255801.
- 31 Maxim Sviridenko. A note on the Kenyon-Remila strip-packing algorithm. *Information Processing Letters*, 112(1-2):10–12, 2012. doi:10.1016/j.ipl.2011.10.003.
- 32 John Turek, Joel L. Wolf, and Philip S. Yu. Approximate Algorithms Scheduling Parallelizable Tasks. In *4th annual ACM symposium on Parallel algorithms and architectures (SPAA)*, pages 323–332, 1992. doi:10.1145/140901.141909.
- 33 Andreas Wiese. A $(1+\epsilon)$ -Approximation for Unsplittable Flow on a Path in Fixed-Parameter Running Time. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 67:1–67:13, 2017. doi:10.4230/LIPIcs.ICALP.2017.67.

Odd-Cycle Separation for Maximum Cut and Binary Quadratic Optimization

Michael Jünger

University of Cologne, Germany
mjuenger@informatik.uni-koeln.de

Sven Mallach

University of Cologne, Germany
mallach@informatik.uni-koeln.de

Abstract

Solving the NP-hard Maximum Cut or Binary Quadratic Optimization Problem to optimality is important in many applications including Physics, Chemistry, Neuroscience, and Circuit Layout. The leading approaches based on linear/semidefinite programming require the separation of so-called odd-cycle inequalities for solving relaxations within their associated branch-and-cut frameworks. In their groundbreaking work, F. Barahona and A.R. Mahjoub have given an informal description of a polynomial-time separation procedure for the odd-cycle inequalities. Since then, the odd-cycle separation problem has broadly been considered solved. However, as we reveal, a straightforward implementation is likely to generate inequalities that are not facet-defining and have further undesired properties. Here, we present a more detailed analysis, along with enhancements to overcome the associated issues efficiently. In a corresponding experimental study, it turns out that these are worthwhile, and may speed up the solution process significantly.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Mathematics of computing → Linear programming; Mathematics of computing → Integer programming

Keywords and phrases Maximum cut, Binary quadratic optimization, Integer linear programming

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.63

Funding *Michael Jünger:* We gratefully acknowledge the funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 764759 *MINOA Mixed-Integer Nonlinear Optimisation: Algorithms and Applications.*

1 Introduction

There are various applications that require solving the Maximum Cut (henceforth MaxCut) problem to optimality, and this has been achieved in the literature by specialized branch-and-cut algorithms that are based on certain relaxations of MaxCut. A prominent example is the determination of ground states in Ising Spin Glasses [1, 3, 4]. Other applications occur in, e.g., Chemistry, Neuroscience, and Circuit Layout. Also, the generic Binary Quadratic Optimization problem (BQP) has a direct transformation to MaxCut, see, e.g. [1], such that enhanced branch-and-cut algorithms for MaxCut directly lead to enhanced branch-and-cut algorithms for the BQP.

A key element to be carried out repeatedly in the course of such branch-and-cut algorithms is the separation of *odd-cycle inequalities* associated with the *cut polytope* as defined in Section 2. In their groundbreaking work, F. Barahona and A.R. Mahjoub [2] have shown that, under certain conditions, the odd-cycle inequalities induce maximal faces (facets) of the cut polytope – which is desirable for their application within a branch-and-cut algorithm. Moreover, they have given an informal description of a polynomial-time separation



© Michael Jünger and Sven Mallach;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 63; pp. 63:1–63:13



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

procedure for these inequalities. Since then, this “algorithmic frame” has been used in many computational studies, but we are not aware of any accounts on the details of the respective implementations or experimental evaluations.

As we show in this paper, however, the odd-cycle inequalities derived using a “straightforward” implementation of this frame are frequently *not* facet-inducing – for two reasons that are both inherent to the algorithm. At the same time, it is, to the best of our knowledge, the only polynomial-time one proposed so far. Along with a more detailed analysis of odd-cycle separation, we thus provide extensions to overcome these shortcomings efficiently by extending the original algorithm. Finally, we present an experimental study showing the practical impact of these enhancements using established benchmark instances for MaxCut and the BQP. It turns out that the additional effort invested is typically more than compensated, i.e., has a positive effect on the solution process.

The outline of this paper is as follows: In Section 2, we define the separation problem for odd-cycle inequalities in the context of solving MaxCut by branch-and-cut, present the outline of the polynomial time separation algorithm given in [2], and point out the shortcomings of a naïve implementation. In Section 3, we analyze the problem formally and present various strategies to enhance the separation procedure. These are experimentally evaluated in Section 4.

2 The Maximum Cut Problem and Odd-Cycle Inequalities

Let $G = (V, E)$ be a simple undirected graph, i.e., there are no loops and no parallel edges. For $k > 1$, a *walk* in G is a set of edges $W = \{\{v_0, v_1\}, \{v_1, v_2\}, \dots, \{v_{k-1}, v_k\}\} \subseteq E$. If, in addition, the vertices $v_0, v_1, \dots, v_k \in V$ are pairwise different, W is called a *path*. Provided that W is a walk and $\{v_0, v_k\} \in E$, we call $W \cup \{v_0, v_k\}$ a *closed walk*. If P is a path, and $\{v_0, v_k\} \in E$, then $C = P \cup \{v_0, v_k\}$ is called a (*simple*) *cycle*. We will also refer to a closed walk that is not a simple cycle as a *non-simple cycle*.

For $W \subseteq V$, let $\delta(W) := \{e = \{u, v\} \in E \mid u \in W, v \in V \setminus W\}$. The edge subsets $\delta(W)$ for any $W \subseteq V$ are the *cuts* of G . They correspond to bipartitions of the vertex set V into W and $V \setminus W$. Given weights w_e for $e \in E$, the maximum cut problem, formulated as an integer linear programming problem, reads:

$$\max \sum_{e \in E} w_e x_e \quad (1)$$

$$\sum_{e \in Q} x_e - \sum_{e \in C \setminus Q} x_e \leq |Q| - 1 \quad \text{for all } Q \subseteq C \subseteq E, C \text{ cycle and } |Q| \text{ odd} \quad (2)$$

$$0 \leq x_e \leq 1 \quad \text{for all } e \in E \quad (3)$$

$$x_e \in \mathbb{Z} \quad \text{for all } e \in E \quad (4)$$

Any solution \hat{x} of (2)–(4) corresponds to a cut $\hat{F} = \{e \in E \mid \hat{x}_e = 1\}$ and vice versa. The necessary and sufficient condition for this to hold is that \hat{F} intersects with every cycle in G in an even number of edges which is enforced by the *odd-cycle inequalities* (2) while the trivial constraints (3) give lower and upper bounds. Consequently, any optimum solution x^* of (2)–(4) gives rise to an optimum cut $F^* = \{e \in E \mid x_e^* = 1\}$ with maximum total weight. We refer to \hat{x} and x^* as the characteristic vectors of \hat{F} and F^* , respectively. Also, we will call a cycle associated with an odd-cycle inequality “odd cycle” although $|C|$ may be even, and refer to it as a pair (C, Q) if its unique determination matters.

The *cut polytope* is the convex hull of the feasible solutions to the maximum cut problem associated with $G = (V, E)$:

$$P_{\text{CUT}}(G) = \text{conv}\{x \in \mathbb{R}^E \mid x \text{ satisfies (2)–(4)}\}$$

At the core of any linear programming based branch-and-cut algorithm, a sequence of linear programming relaxations of (1)–(3) is solved by a *cutting plane algorithm*: The first relaxation just consists of (1) and (3). Then it is the task of an *odd-cycle separation algorithm* to decide if the optimum solution \hat{x} of the current relaxation satisfies all odd-cycle constraints (2). If so, the linear programming relaxation (1)–(3) is solved to optimality and its objective function value is used as an upper bound in a branch-and-bound scheme. If not, the separation algorithm must provide at least one odd-cycle inequality that is violated by \hat{x} . The produced inequalities are then added to strengthen the current relaxation, this new relaxation is solved to optimality, and the process is iterated until the separation algorithm decides that the solution of the current linear program satisfies all odd-cycle constraints.

In branch-and-cut, it is preferred that the separated inequalities define facets of the polytope associated with the problem, so here, we would prefer odd-cycle inequalities that define facets of the cut polytope $P_{\text{CUT}}(G)$. Barahona and Mahjoub [2] have given a proof that an odd-cycle inequality defines a facet of $P_{\text{CUT}}(G)$ if and only if its associated cycle C is *chordless*, i.e., if there is no edge in $E \setminus C$ that connects two vertices of C .

2.1 Polynomial-Time Separation of Odd-Cycle Inequalities

We describe the polynomial time algorithm of Barahona and Mahjoub [2]. For this purpose, we rewrite the odd-cycle inequalities (2) associated with a graph $G = (V, E)$ as follows:

$$\sum_{e \in Q} (1 - x_e) + \sum_{e \in C \setminus Q} x_e \geq 1 \quad \text{for all } Q \subseteq C \subseteq E, C \text{ cycle and } |Q| \text{ odd} \quad (5)$$

Our task is to solve the separation problem for $\hat{x} \in [0, 1]^E$. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two copies of G , and denote with $u_1 \in V_1$ the representative of $u \in V$ in G_1 and with $u_2 \in V_2$ the one in G_2 . Based on these copies, define a new weighted graph $G_S = (V_S, E_S)$ with $V_S = V_1 \cup V_2$, $E_S = E_1 \cup E_2 \cup E_3$. The additional edge set $E_3 \subseteq V_1 \times V_2$ consists of the edges $\{u_1, v_2\}$ and $\{v_1, u_2\}$ for each $\{u, v\} \in E$. The edges $\{u_1, v_1\} \in E_1$ receive weights $\omega(\{u_1, v_1\}) = \hat{x}_{\{u, v\}}$, the edges $\{u_2, v_2\} \in E_2$ also receive weights $\omega(\{u_2, v_2\}) = \hat{x}_{\{u, v\}}$, and the edges $\{u_1, v_2\}, \{v_1, u_2\} \in E_3$ receive “inverted” weights $\omega(\{u_1, v_2\}) = \omega(\{v_1, u_2\}) = 1 - \hat{x}_{\{u, v\}}$. To ease notation, we will frequently write $\omega(F) := \sum_{e \in F} \omega(e)$ for any edge set $F \subseteq E_S$ or $F \subseteq E$. Moreover, we will also write $V_S(F)$ or $V(F)$ to denote the vertex set $\{v \in V_S \mid \exists \{v, w\} \in F\}$ or $\{v \in V \mid \exists \{v, w\} \in F\}$, respectively.

The fundamental property of the construction described is that, for any $u \in V$, any path $P_u \subseteq E_S$ from u_1 to u_2 in G_S corresponds to a closed walk C in the original graph G containing u . Moreover, it inevitably involves an odd number of edges from the set E_3 that, as the associated ω -weights indicate, define the subset $Q \subseteq C$.

Thus, each such path P_u corresponds to an odd closed walk (C, Q) in G that may however have vertex as well as edge repetitions, i.e., C is not necessarily a simple cycle. Fig. 1 shows a small example where two different paths of equal length (w.r.t. ω as well as the total number of edges) in G_S lead to either a simple or non-simple cycle in G .

Now, the central idea of the algorithm is to compute, for each $u \in V$, an ω -shortest path P_u from u_1 to u_2 in G_S .

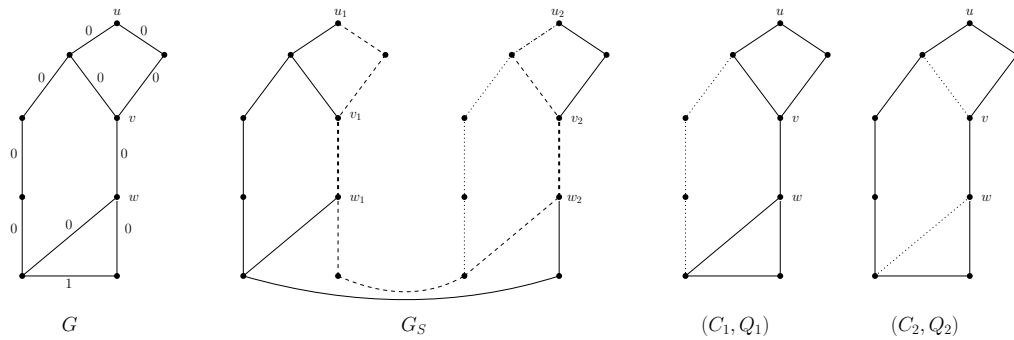
If then $\omega(P_u) \geq 1$ for every $u \in V$, then there is no odd closed walk (C, Q) that violates

$$\sum_{e \in Q} (1 - x_e) + \sum_{e \in C \setminus Q} x_e \geq 1 \quad \text{for all } Q \subseteq C \subseteq E, C \text{ closed walk and } |Q| \text{ odd} \quad (6)$$

and, therefore, no simple cycle C that violates (5).

Otherwise, if $\omega(P_u) < 1$ for some $u \in V$, then P_u corresponds to an odd closed walk (C, Q) that violates (6). However, only if (C, Q) is “accidentally” simple, we have found a violated inequality of type (5).

This does not affect the correctness of the approach: Even when using (C, Q) “as is” in the cutting plane algorithm, i.e., when possibly adding an inequality of type (6) rather than one of type (5) to the linear program, the final result satisfies all the latter and thus solves the relaxation (1)–(3) of MaxCut. However, while inequalities (6) are valid for $P_{\text{CUT}}(G)$, they are not facet-inducing. And even if a cycle (C, Q) derived from an ω -shortest path in G_S is simple, it is very likely to have chords as we show in Sect. 3.3. So in summary, when using the described approach, the likelihood to generate cutting planes that do *not* correspond to facets of $P_{\text{CUT}}(G)$ is high. Fortunately, we can efficiently construct from P_u a simple odd cycle (C, Q) that violates (5), and even one that is chordless, as we shall see in Sect. 3.2 and 3.3, respectively.



■ **Figure 1** Left: An example graph G whose edges are annotated with an (integral) linear program solution. Middle: The corresponding separation graph G_S (only zero-weight edges are shown). The dashed edges correspond to a shortest u_1 - u_2 -path that induces a non-simple odd closed walk in G (edge $\{v, w\}$ is used twice, see the left of the two cycles depicted on the right). If the subpath within the right copy of G is replaced by the dotted one, the induced cycle is simple (but not chordless).

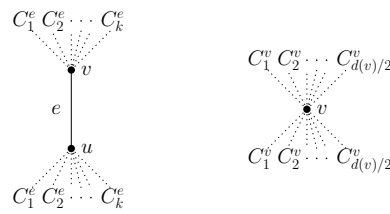
3 Non-Simple and Non-Chordless Simple Odd Cycles

To the best of our knowledge, the possible non-simplicity of odd closed walks derived with the common procedure, as well as the possible presence of chords, has attracted almost no further consideration in the literature before. However, as we will see, it matters not only in theory but also in practice.

Besides the preserved correctness of the separation algorithm already discussed, another possible reason why non-simplicity has been ignored may be the existence of an easy proof that any odd closed walk (C, Q) contains at least one simple cycle (C', Q') with $C' \subseteq C$, $Q' \subseteq Q$, and with $|Q'|$ odd:

If (C, Q) is non-simple, then either there is at least one edge $e = \{u, v\} \in E$ that occurs $k > 1$ times on C . Or, each edge $e \in E$ occurs at most once on C , but there is a vertex $v \in V$ whose degree w.r.t. the edges of C is $d_C(v) := |\{w \in V : \{v, w\} \in C\}| > 2$ (cf. Fig. 2). Since we assumed before that no edge is contained twice in C , $d_C(v)$ must then be even¹.

¹ The present case of odd closed walks induced by simple paths P in G_S with weight $\omega(P) < 1$ implies $k \leq 2$ since any combination of an E_3 -instance with an E_1 - or E_2 -instance of the same edge $e \in E$ leads to an ω -weight of at least one. Moreover, the simplicity of P_u implies $d_C(v) \leq 4$ for all $v \in V$.



■ **Figure 2** Multiple cycles comprising an edge $\{u, v\} \in E$ or a single vertex $v \in V$.

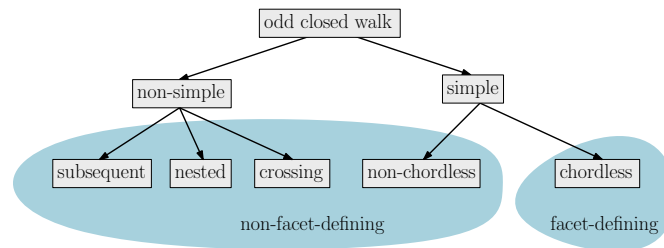
In the first case, let C_1^e, \dots, C_k^e be the corresponding closed walks that result from starting a path in v – each time using a different edge that is not e – and continuing the path until v is reached again first using e . In the second case, let $C_1^v, \dots, C_{\frac{d_C(v)}{2}}^v$ be the respective closed walks that result from starting a path in v using each time a different edge, and ending when v is first reached again.

We refer to the associated subsets of Q as $Q_i^e := C_i^e \cap Q, i \in \{1, \dots, k\}$, and $Q_i^v := C_i^v \cap Q, i \in \{1, \dots, \frac{d_C(v)}{2}\}$, respectively. Since Q is odd, there is at least one $i^* \in \{1, \dots, k\}$ ($i^* \in \{1, \dots, \frac{d_C(v)}{2}\}$) such that $|Q_{i^*}^e| \leq |Q|$ ($|Q_{i^*}^v| \leq |Q|$) is odd as well. Now either $(C_{i^*}^e, Q_{i^*}^e)$ ($(C_{i^*}^v, Q_{i^*}^v)$) is simple in which case we are done. Or, we find an edge $f \in C_{i^*}^e, f \neq e$, that occurs more than once on $C_{i^*}^e$ (a vertex $w \neq v$ in $C_{i^*}^e$ such that $d_{C_{i^*}^e}(w) > 2$) and have thus reduced the extraction of a simple cycle to a strictly smaller non-simple one. Thus, since any simple cycle has length at least three, the according recursion must terminate after finitely many steps with a cycle that is simple.

It however remains open how to *extract* a (chordless) simple odd closed walk from a non-simple one *efficiently*. This will be addressed in Sect. 3.2 and Sect. 3.3 after clarifying in Sect. 3.1 which kinds of non-simplicity can occur from the method described in Sect. 2.1.

3.1 Cases of Non-Simplicity

The landscape of different cases and their implications concerning the facet-defining property of the associated odd-cycle inequalities is depicted in Fig. 3.

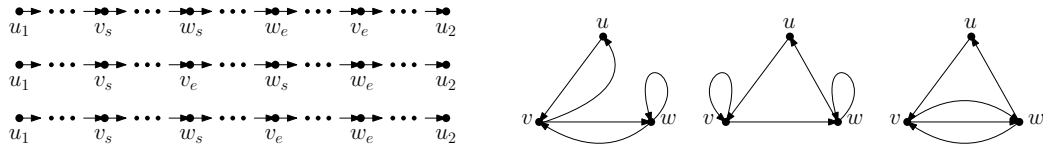


■ **Figure 3** Categorization of odd closed walks.

Let P_u be an ω -shortest path from u_1 to u_2 in G_S , and let (C, Q) be its induced odd closed walk in G . We may assume w.l.o.g. (and enforce in an implementation) that P_u does not enter u_1 and that it does not leave u_2 which implies that $d_C(u) = 2$.

Now if (C, Q) is non-simple, there must be at least one vertex v with $d_C(v) > 2$, i.e., $v \neq u$. The simplicity of P_u then implies that it must traverse v_1 as well as v_2 . Denote the corresponding subpath, that itself defines an odd closed walk, with P_v . It is *fully nested* within P_u , i.e., $P_v \subsetneq P_u$. Moreover, since the number of E_3 -edges in P_u as well as in P_v is odd, the number of E_3 -edges in $P_u \setminus P_v$ must be even. Hence, while its induced edge set in G is still a closed walk, it is not an odd one.

Of course, an odd closed walk (C, Q) induced by an ω -shortest path P_u in G_S may have not only one but several vertices $v \in V$ with $d_C(v) > 2$. If all the associated subpaths of P_u are fully nested, the previous arguments imply that only the innermost one of them induces a simple odd cycle. Otherwise, any two strict subpaths of P_u , say P_v and P_w , may either occur *subsequently*, i.e., $P_v \cap P_w = \emptyset$, or *crossing*, i.e., either w_1 or w_2 occurs in P_v but not both. Fig. 4 displays the respective vertex sequences using the notation v_s and v_e (for “start” and “end”) since, for any $v \in V$, $v \neq u$, v_2 may be visited by P_u before v_1 is visited and vice versa.



■ **Figure 4** Left: Schematic depiction of (from top to bottom) two fully nested, subsequent, and crossing subpaths P_v and P_w of P_u . Right: The corresponding interpretations in the original graph (from left to right).

In the subsequent case, both P_v and P_w correspond to odd closed walks since they both comprise an odd-cardinality subset of E_3 -edges (but neither $P_u \setminus P_v$ nor $P_u \setminus P_w$ does). In the crossing case, P_v and P_w both correspond to odd closed walks as well. However, if the subpaths from v_s to w_s and from v_e to w_e refer to the same edge set in G , then both odd closed walks associated to P_v and P_w are equivalent (and thus lead to the same inequality).

3.2 Handling Non-Simple and Chorded Simple Odd Cycles

There are basically three ways to deal with non-simplicity.

3.2.1 The “Take-As-Is” Strategy

Since the potential non-simplicity of a closed odd walk (C, Q) does not affect the validity of the associated inequality, one may consider to simply *use it as a cutting plane as is*.

3.2.2 The “Throw-Away” Strategy

Another strategy is to *discard* non-simple closed odd walks, i.e., to generate an inequality only if a closed odd walk computed is “accidentally” simple.

However, to still preserve an *exact* separation algorithm, it must then be sure that *at least one* simple cycle is still identified if a violated odd-cycle inequality exists. Moreover, as already discussed in Sect. 2.1, any vertex $u \in V$ may have several ω -shortest paths P_u some of which may induce non-simple cycles, and that may even all have the same number of edges. So “throwing away” any such P_u containing a subpath P_v by “trusting” that a simple cycle will be identified when computing an ω -shortest v_1 - v_2 -path may fail in general.

Fortunately, the desired guarantee can nevertheless be established if we strive for (ω, ℓ) -shortest paths instead, where $\ell(v) \in \mathbb{Z}$ denotes the number of edges used in a (currently) shortest path to each vertex $v \in V_S$. In this context, a u_1 - v -path P in G_S involving edge $\{v, w\} \in E_S$ is (ω, ℓ) -shorter than the best previous one, if either “classically” $\text{dist}(u) + \omega(u, v) < \text{dist}(v)$, or if $\text{dist}(u) + \omega(u, v) = \text{dist}(v)$ and $\ell(u) + 1 < \ell(v)$. With these extensions, the original separation algorithm will identify at least one violated odd-cycle inequality if any exists, as we will now show.

► **Theorem 1.** *Let $P_u \subseteq E_S$ be an ω -shortest path from u_1 to u_2 in G_S with weight $\omega(P_u) < 1$ and of edge length $\ell(P_u)$. Then there exists a vertex $w \in V$ such that each (ω, ℓ) -shortest path P_w in G_S satisfies $\omega(P_w) \leq \omega(P_u)$, $|P_w| \leq |P_u|$, and whose induced odd closed walk (C, Q) in G is simple.*

Proof. If the odd closed walk induced by P_u is itself simple, the statement follows by choosing $P_w = P_u$. So suppose this is not the case. Then there exists a strict subpath P_v of P_u . Now either P_v is simple, in which case we may choose $P_w = P_v$. Otherwise, we recurse on the path P_v that clearly satisfies $|P_v| \leq |P_u| - 2$. Since the number of edges of the paths considered thus decreases strictly monotonically, and since any shortest path in G_S has a positive length, this process will eventually terminate at some simple path P_w , $w \in V_S$, as otherwise, there would exist a path of the same ω -weight which is shorter w.r.t. ℓ – a contradiction. ◀

3.2.3 The “Extraction” Strategy

A final strategy to deal with non-simple odd closed walks is to extract the simple odd cycles contained in it. This is motivated by several advantages, e.g.:

- If a shortest inner subpath $P_v \subseteq P_u$ corresponding to a simple cycle in G is found, the shortest path computation w.r.t. v may be omitted (accepting a possible loss of a different inequality corresponding to another shortest v_1 - v_2 -path).
- A single pass of P_u might allow to extract *several* simple cycles from one non-simple one.
- Information found during the extraction might be used to avoid the addition of duplicate inequalities (cf. the case of crossing subpaths at the end of Sect. 3.1).

Moreover, as expressed in the following observation, it is possible to implement simple cycle extractions requiring only a number of operations that is proportional to the length of the closed walk (which can be no more than $2|V|$). So since a *single pass* of the closed walk is inevitably required to even generate an inequality from it or find out whether it is simple, the respective enhancement does not increase the asymptotic running time, and does not even require the (ω, ℓ) -extension from Sect. 3.2.2 (which is nevertheless worthwhile) in order to provide a safe *exact* separation procedure.

► **Observation 2.** *Let P_u be an ω -shortest path from u_1 to u_2 in G_S with weight $\omega(P_u) < 1$, and whose induced odd closed walk (C, Q) in G is not simple. Traverse P_u starting from u_1 (or, walking backwards, from u_2) and let $v \in W$ be the first vertex such that v_1 and v_2 were both visited on this traversal. Then the cycle in G corresponding to the path P_v is simple.*

An inequality derived based on Observation 2 (i.e., on a first *fully nested* subpath P_v of P_u that does not contain any further fully nested subpaths itself) neither needs to be the only one that can be extracted from P_u (cf. Sect. 3.1), nor needs to be maximally violated among these.

Fortunately, subsequent and crossing simple cycles may as well be easily recognized algorithmically by only maintaining the positions of the start and end vertices of the *last* subpath identified as to correspond to a simple cycle. Suppose that P_v was the last such subpath on a traversal of P_u , i.e., the positions of v_s and v_e are known. Assume further that the traversal then arrives at some vertex, say w_e , whose other copy, w_s , has already been visited. If $w_s \prec v_s$, then P_v is fully nested within P_w , and thus P_w need not be considered. Otherwise, either $w_s \prec v_e$ in which case P_v and P_w cross, or $v_e \prec w_s$ in which case P_w is subsequent to P_v (cf. Fig. 4). In any case, we can safely update the two maintained positions to those of w_s and w_e . We will then classify later paths correctly since a later path crossing P_v must inevitably either also cross P_w or contain P_w fully nested.

Additionally, a constant-time check to eliminate some unwanted duplicate inequalities in the case of crossing paths P_v and P_w is right at hand from the final statement in Sect. 3.1. If v_e is the immediate predecessor of w_e on P_u , then the same is likely true (sure if computing (ω, ℓ) -shortest paths) for v_s and w_s . This means that $\{v, w\}$ is a chord (being part of the odd closed walk w.r.t. P_u), and the odd-cycle inequalities according to P_v and P_w are equivalent.

3.2.4 Further Engineering of the Extraction Strategy

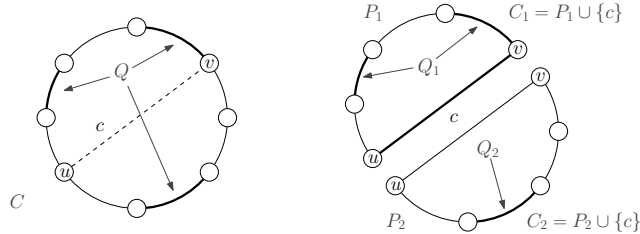
A further opportunity to improve the practical performance of the separation procedure is to exploit the symmetry of the graph G_S : Suppose that, while computing a shortest u_1 - u_2 -path in G_S , we process a vertex v_2 (v_1) such that the distances $\text{dist}(v_1)$ and $\text{dist}(v_2)$ are both known. If $\text{dist}(v_1) + \text{dist}(v_2) \geq 1$, then v_2 (v_1) cannot lie on any u_1 - u_2 -path that induces a violated odd-cycle inequality and thus need not be considered for the update of other distances.

3.3 Chorded Simple Odd Cycles

Finally, as we now know how to extract simple odd cycles, it is natural to strive for chordless ones, i.e. facet-defining inequalities. However, we will show in this subsection, why we will not always find these when searching for *maximally violated* inequalities (as we do when computing ω - or (ω, ℓ) -shortest paths).

For this purpose, let (C, Q) be a simple cycle, and suppose that, for some $\hat{x} \in [0, 1]^E$, the associated odd-cycle inequality is violated, i.e. $\sum_{e \in Q} (1 - \hat{x}_e) + \sum_{e \in C \setminus Q} \hat{x}_e < 1$.

Suppose further that C has a chord c , i.e., there exist vertices $u, v \in V$ such that $c = \{u, v\} \in E \setminus C$. Partitioning C into the corresponding two $\{u, v\}$ -paths P_1 and P_2 partitions Q into Q_1 and Q_2 , and inevitably renders either $|Q_1|$ odd and $|Q_2|$ even or vice versa (cf. Fig. 5).



■ **Figure 5** Left: A cycle C with an odd-cardinality edge subset Q (thicker), assumed to have a chord $c = \{u, v\}$. Right: Splitting C into two chordless odd cycles C_1 and C_2 , declaring c once as “odd” and once as “even”.

Assume w.l.o.g. that $|Q_1|$ is even. The cycles $C_1 = P_1 \cup \{c\}$, with c “marked as odd”, and $C_2 = P_2 \cup \{c\}$, with c “marked as even”, correspond to the two odd-cycle inequalities

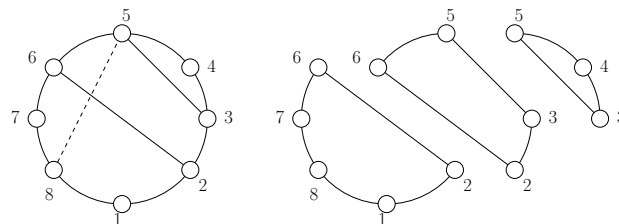
$$\sum_{e \in Q_1} (1 - x_e) + (1 - x_c) + \sum_{e \in P_1 \setminus Q_1} x_e \geq 1 \tag{7}$$

$$\sum_{e \in Q_2} (1 - x_e) + x_c + \sum_{e \in P_2 \setminus Q_2} x_e \geq 1 \tag{8}$$

whose sum gives the original one above assumed to be violated. This implies that at least one of (7) and (8) needs to be violated as well.

Let $\gamma_1 := 1 - \sum_{e \in Q_1} (1 - \hat{x}_e) - (1 - \hat{x}_c) - \sum_{e \in P_1 \setminus Q_1} \hat{x}_e$ and $\gamma_2 := 1 - \sum_{e \in Q_2} (1 - \hat{x}_e) - \hat{x}_c - \sum_{e \in P_2 \setminus Q_2} \hat{x}_e$ denote the violations of (7) and (8), respectively. Then $\gamma := \gamma_1 + \gamma_2$ is the violation of the original inequality. Therefore, one of (7) or (8) can only be found if the other is not violated at all.

It remains to clarify whether we can *efficiently* extract chordless cycles from a simple but not chordless one. This turns out to be the case when restricting to *one* chordless cycle whose associated inequality is violated. In this case, we may use any chord to (conceptually) split the initial cycle (C, Q) into two cycles, and proceed with these in the same fashion if necessary. This can be implemented in a way such that no chord is used for a split more than once, and the adjacency list of each vertex $v \in V(C)$ is traversed at most once (by continuing at the list position of a possible previous visit of the vertex, cf. Fig. 6). Assuming the presence of the respective data structures, and the use of bucket sort to order the adjacency lists appropriately, an asymptotic running time of $\mathcal{O}(|E|)$ can be achieved.



■ **Figure 6** Decomposition of a simple cycle with chords to retrieve *one* chordless cycle. After numbering the vertices consecutively and sorting adjacency lists descendingly w.r.t. this numbering, the decomposition can be carried out in a greedy fashion. If, e.g., the inequality associated to the first chordless cycle depicted on the right turns out not to be violated, it can be neglected by backtracking to vertex 2 and continuing with the next edge of its adjacency list. Even if further subdecompositions take place, the process will never consider any edge more than twice. In particular, no enumerative consideration of chords used as split edges is required – e.g., the chordless cycles related to the chord $\{5, 8\}$ crossing the cycles considered before can be neglected.

4 Experiments

Our experimental study shall particularly address the following two questions:

- Does the “quality” of odd-cycle cutting planes matter, i.e., to what extent is the process of solving the linear programming relaxations of maximum cut and binary quadratic optimization problems affected by the exclusive separation of odd-cycle inequalities that correspond to simple or even chordless simple cycles?
- Is it worthwhile to invest time for the extraction of (chordless) simple cycles (and to save some shortest path computations) rather than to simply employ or discard non-simple cycles?

To this end, we implemented an odd-cycle separation algorithm supporting the “take-as-is”-strategy, the “throw-away”-strategy, the extraction of simple cycles, an improved extraction exploiting the symmetry of the separation graph as described in Sect. 3.2.4, and finally the extraction of one chordless cycle from each simple one. In each of these variants, (ω, ℓ) -shortest paths are computed for each $v \in V$ in general. The three mentioned extraction variants however omit the shortest path computation w.r.t. a vertex $v \in V$ if a simple cycle based on a shortest subpath P_v has been identified before.

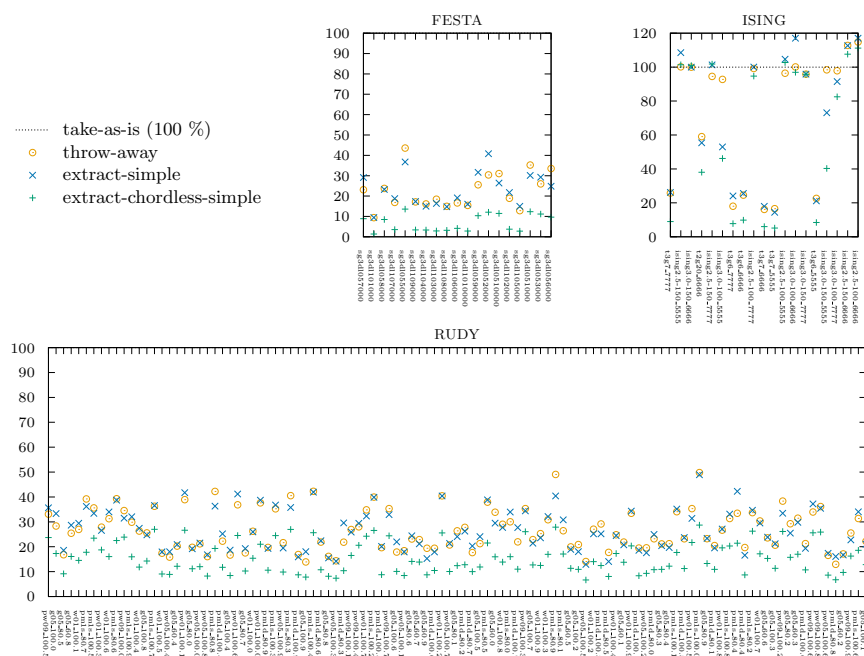
Based on this implementation, we solved the linear programming relaxations of several instances from the “binary quadratic and max cut library” [5] by iteratively calling the linear program solver Gurobi² in version 8, and our separation algorithm. In case of the MaxCut instances, the initial linear program defined only lower and upper bounds of the variables. In case of BQP instances, we started with all inequalities that correspond to the respective standard linearization. Moreover, after solving a linear program, previously added inequalities not being satisfied at equality, were removed.

For presentation, we selected those instances where *at least one cut was required to solve the relaxation* and the *total time needed for this was at most one hour and at least half a second*.

4.1 Odd Cycle Separation Quality

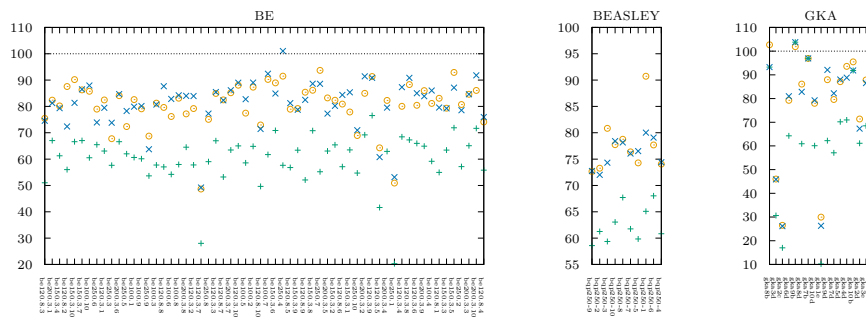
The first two experiments relate to the first question mentioned in the beginning of this section. As a first quality indicator, we consider *the total number of inequalities added* while solving the relaxation and when using the different strategies. Since however, a smaller number of cuts might result in an increase of the required number of linear programs to solve, and there is a natural decrease in the number of generated cuts *per iteration* of the extraction and “take-as-is”-strategies compared to the “throw-away”-strategy, we complement this evaluation with another one that considers *the total relaxation solution time*.

In both experiments, the “take-as-is”-strategy serves as the basis (100% level in the figures), and the symmetry-aware version of the extraction strategy is omitted from the presentation, as it generates the same inequalities as the one without symmetry-awareness. Moreover, to reduce side-effects, duplicate inequalities found during any of the variants are not counted and eliminated before passing the cuts to the linear program.

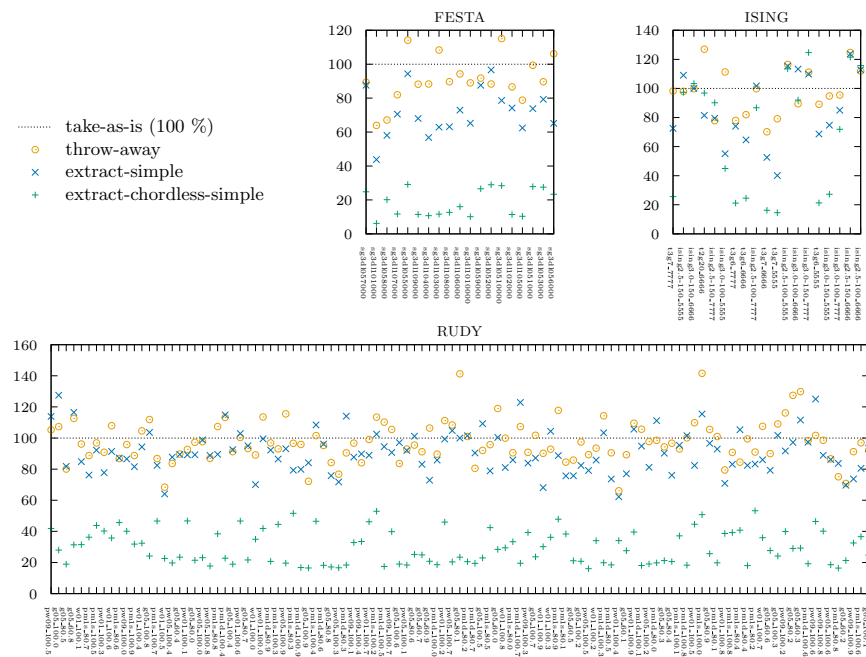


■ **Figure 7** Relative total number of inequalities added (MaxCut instances).

² <http://www.gurobi.com/>



■ **Figure 8** Relative total number of inequalities added (BQP instances).



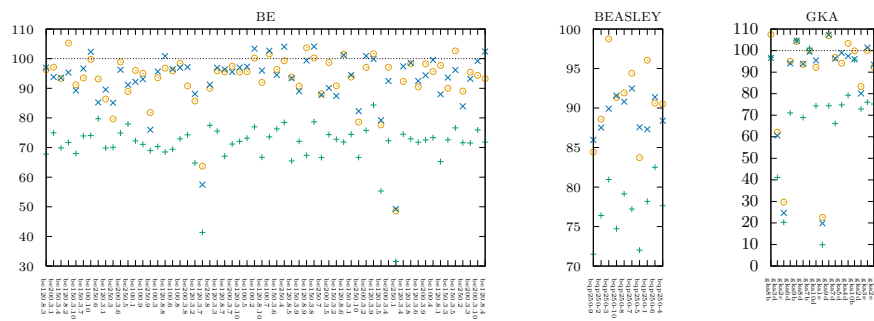
■ **Figure 9** Relative total relaxation solution time (MaxCut instances).

Figs. 7 and 8 show a clear trend of decrease of the total number of inequalities added when restricting to simple or even only chordless simple cycles. Indeed the quality of a cut appears to have a significant impact, as the extraction of the facet-defining chordless simple cycles clearly performs best on average across the instances considered. Moreover, in Figs. 9 and 10, one can see that, in most of the cases, the additional effort to extract simple or one chordless simple cycle does not lead to an increase but a noticeable decrease of the total relaxation solution time. This means in particular that the advantage of requiring fewer cuts is a true one, i.e., not negatively compensated by a significant increase of the number of linear programs required to solve the relaxation.

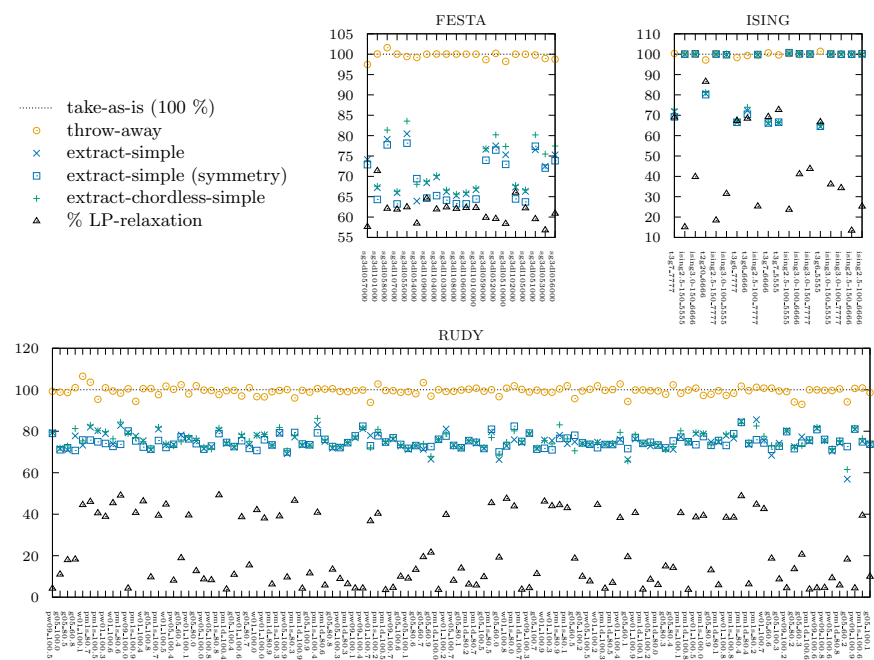
4.2 Odd Cycle Separation Time

The total relaxation solution time considered in the last experiment is affected by several aspects. In particular, the different runs per instance did not correspond to the same series of linear programs, as different cuts lead to different solutions. To address the second

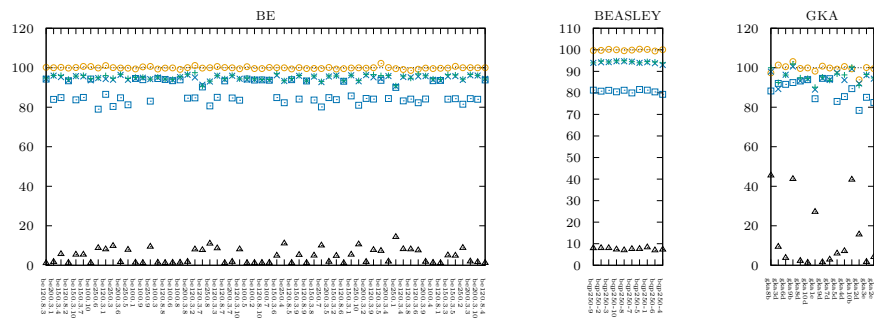
63:12 Odd-Cycle Separation for MaxCut and the BQP



■ Figure 10 Relative total relaxation solution time (BQP instances).



■ Figure 11 Relative accumulated separation times (MaxCut instances).



■ Figure 12 Relative accumulated separation times (BQP instances).

question mentioned at the beginning of this section, i.e., in order to show that simple and even chordless simple cycle extraction truly come at negligible cost – we now create equal preconditions for all separation strategies as follows: After solving a linear program, the separator is called four times, once with each of the different strategies. After that, *one* cutting plane set (the inequalities generated in the “extract-simple”-run) is added to the linear program and the procedure is iterated.

In this experiment, we thus consider the *accumulated time spent in the separation procedure* over all linear programs solved in order to solve the respective relaxation, with the presentation restricted to those instances where this was *at least half a second*. As another difference to the previous experiment, here, duplicate inequalities are not avoided in general. In case of the extraction strategies, they are however avoided in the way described in Sect. 3.2.3.

The results are shown in Figs. 11 and 12, as well as the percentage of time spent in separation w.r.t. the total time to solve the relaxation in the fastest of the four cases (denoted “% LP-relaxation”). As before, the “take-as-is”-strategy (100% level) serves as the basis for comparisons. Across almost all the instances considered, the extraction of simple cycles results in a quicker separation compared to the “take-as-is”- and the “throw-away”-strategy. Not surprisingly, the latter two perform very similarly, as both require a single pass of each identified odd closed walk. Exploiting symmetries usually gives another measurable speedup. This may of course as well be applied to the chordless extraction variant which has anyway not been implemented as efficiently as indicated in Sect. 3.3.

References

- 1 Francisco Barahona, Michael Jünger, and Gerhard Reinelt. Experiments in quadratic 0–1 programming. *Mathematical Programming*, 44(1):127–137, May 1989. doi:10.1007/BF01587084.
- 2 Francisco Barahona and Ali Ridha Mahjoub. On the cut polytope. *Mathematical Programming*, 36(2):157–173, June 1986. doi:10.1007/BF02592023.
- 3 Caterina De Simone, Martin Diehl, Michael Jünger, Petra Mutzel, Gerhard Reinelt, and Giovanni Rinaldi. Exact ground states of Ising spin glasses: New experimental results with a branch-and-cut algorithm. *Journal of Statistical Physics*, 80(1-2):487–496, 1995.
- 4 Caterina De Simone, Martin Diehl, Michael Jünger, Petra Mutzel, Gerhard Reinelt, and Giovanni Rinaldi. Exact ground states of two-dimensional $\pm J$ Ising spin glasses. *Journal of Statistical Physics*, 84(5):1363–1371, September 1996. doi:10.1007/BF02174135.
- 5 Franz Rendl, Giovanni Rinaldi, and Angelika Wiegele. A Branch and Bound Algorithm for Max-Cut Based on Combining Semidefinite and Polyhedral Relaxations. In Matteo Fischetti and David P. Williamson, editors, *Integer Programming and Combinatorial Optimization*, pages 295–309, Berlin, Heidelberg, 2007. Springer.

Triangles and Girth in Disk Graphs and Transmission Graphs

Haim Kaplan

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

Katharina Klost

Institut für Informatik, Freie Universität Berlin, 14195 Berlin, Germany
kathklost@inf.fu-berlin.de

Wolfgang Mulzer 

Institut für Informatik, Freie Universität Berlin, 14195 Berlin, Germany
mulzer@inf.fu-berlin.de

Liam Roditty

Department of Computer Science, Bar Ilan University, Ramat Gan 5290002, Israel
liamr@macs.biu.ac.il

Paul Seifert

Institut für Informatik, Freie Universität Berlin, 14195 Berlin, Germany
pseifert@inf.fu-berlin.de

Micha Sharir

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

Abstract

Let $S \subset \mathbb{R}^2$ be a set of n sites, where each $s \in S$ has an associated radius $r_s > 0$. The *disk graph* $D(S)$ is the undirected graph with vertex set S and an undirected edge between two sites $s, t \in S$ if and only if $|st| \leq r_s + r_t$, i.e., if the disks with centers s and t and respective radii r_s and r_t intersect. Disk graphs are used to model sensor networks. Similarly, the *transmission graph* $T(S)$ is the directed graph with vertex set S and a directed edge from a site s to a site t if and only if $|st| \leq r_s$, i.e., if t lies in the disk with center s and radius r_s .

We provide algorithms for detecting (directed) triangles and, more generally, computing the length of a shortest cycle (the *girth*) in $D(S)$ and in $T(S)$. These problems are notoriously hard in general, but better solutions exist for special graph classes such as planar graphs. We obtain similarly efficient results for disk graphs and for transmission graphs. More precisely, we show that a shortest (Euclidean) triangle in $D(S)$ and in $T(S)$ can be found in $O(n \log n)$ expected time, and that the (weighted) girth of $D(S)$ can be found in $O(n \log n)$ expected time. For this, we develop new tools for batched range searching that may be of independent interest.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Theory of computation \rightarrow Graph algorithms analysis

Keywords and phrases disk graph, transmission graph, triangle, girth

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.64

Related Version A full version is available on the arXiv (<https://arxiv.org/abs/1907.01980>).

Funding Supported in part by grant 1367/2016 from the German-Israeli Science Foundation (GIF).
Wolfgang Mulzer: Partially supported by ERC STG 757609.

Paul Seifert: Partially supported by DFG grant MU/3501/1.

Micha Sharir: Partially supported by ISF Grant 892/13, the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11), the Blavatnik Research Fund in Computer Science at Tel Aviv University, and the Hermann Minkowski-MINERVA Center for Geometry at Tel Aviv University.

Acknowledgements We like to thank Günther Rote and Valentin Polishchuk for helpful comments.



© Haim Kaplan, Katharina Klost, Wolfgang Mulzer, Liam Roditty, Paul Seifert, and Micha Sharir; licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 64; pp. 64:1–64:14

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Given a graph G with n vertices and m edges, does G contain a *triangle* (a cycle with three vertices)? This is one of the most basic algorithmic questions in graph theory, and many other problems reduce to it [12, 21]. The best known algorithms use fast matrix multiplication and run in either $O(n^\omega)$ time or in $O(m^{2\omega/(\omega+1)})$ time, where $\omega < 2.37287$ is the matrix multiplication exponent [1, 10, 12]. Despite decades of research, the best available “combinatorial” algorithm¹ needs $O(n^3 \text{polyloglog}(n)/\log^4 n)$ time [22], only slightly better than checking all vertex triples. This lack of progress can be explained by a connection to Boolean matrix multiplication (BMM): if there is a truly subcubic combinatorial algorithm for finding triangles, there is also a truly subcubic combinatorial algorithm for BMM [21]. Itai and Rodeh [12] reduced computing the *girth* (the length of a shortest cycle) of an unweighted undirected graph to triangle detection. For integer edge weights, Roditty and V. Williams [19] gave an equivalence between finding a minimum weight cycle (the weighted girth) and finding a minimum weight triangle.

For the special case of *planar* graphs, significantly better algorithms are known. Itai and Rodeh [12] and, independently, Papadimitriou and Yannakakis [17] showed that a triangle can be found in $O(n)$ time, if it exists. Chang and Lu [7] presented an $O(n)$ time algorithm for computing the girth. The weighted girth can be found in $O(n \log \log n)$ time both in an undirected and in a directed planar graph [15, 16].

In computational geometry, there are two noteworthy graph classes that generalize planar graphs: *disk graphs* and *transmission graphs*. We are given a set S of n planar point *sites*. Each $s \in S$ has an *associated radius* $r_s > 0$ and an *associated disk* D_s with center s and radius r_s . The *disk graph* $D(S)$ is the undirected graph on S where two sites $s, t \in S$ are adjacent if and only if D_s and D_t intersect, i.e., $|st| \leq r_s + r_t$, where $|\cdot|$ is the Euclidean distance. In a *weighted disk graph*, the edges are weighted according to the Euclidean distance between their endpoints. The *transmission graph* $T(S)$ is the directed graph on S where there is an edge from s to t if and only if t lies in D_s , i.e., $|st| \leq r_s$. Again, there is a weighted variant. Both graph classes have received a lot of attention, as they give simple and natural theoretical models for geometric sensor networks (see, e.g., [13, 14]).

Motivated by the vastly better algorithms for planar graphs, we investigate triangle detection and girth computation in disk graphs and transmission graphs. We will see that in a disk graph, a triangle can be found in $O(n \log n)$ time, using a simple geometric observation to relate disk graphs and planar graphs. By a reduction from ε -CLOSENESS [18], this is optimal in the algebraic decision tree model, a contrast to planar graphs, where $O(n)$ time is possible. Our method generalizes to finding a shortest triangle in a weighted disk graph in $O(n \log n)$ expected time. Moreover, we can compute the unweighted and weighted girth in a disk graph in $O(n \log n)$ time, with a deterministic algorithm for the unweighted case and a randomized algorithm for the weighted case. The latter result requires a method to find a shortest cycle that contains a given vertex. Finally, we provide an algorithm to detect a directed triangle in a transmission graph in $O(n \log n)$ expected time. For this, we study the geometric properties of such triangles in more detail, and we develop several new techniques for batched range searching that might be of independent interest, using linearized quadtrees and three-dimensional polytopes to test for containment in the union of planar disks. As before, this algorithm extends to the weighted version. We will assume *general position*, meaning that all edge lengths (and more generally shortest path distances) are pairwise

¹ An algorithm is “combinatorial” if it does not need algebraic manipulations to achieve its goal.

distinct, that no site lies on a disk boundary, and that all radii are pairwise distinct. Due to space reasons, some proofs in this extended abstract are only sketched. The complete proofs can be found in the full version of this paper.

2 Finding a (Shortest) Triangle in a Disk Graph

We would like to decide if a given disk graph contains a triangle. If so, we would also like to find a triangle of minimum Euclidean perimeter.

2.1 The Unweighted Case

The following property of disk graphs, due to Evans et al. [9], is the key to our algorithm. For a proof refer to the full version.

► **Lemma 2.1.** *Let $D(S)$ be a disk graph that is not plane, i.e., the embedding that represents each edge by a line segment between its endpoints has two segments that cross in their relative interiors. Then, there are three sites whose associated disks intersect in a common point.*

If $D(S)$ is not plane, it contains a triangle by Lemma 2.1. If $D(S)$ is plane, we can construct it explicitly and then search for a triangle in $O(n)$ time [12, 17]. To check whether $D(S)$ is plane, we begin an explicit construction of $D(S)$ and abort if we discover too many edges.

► **Theorem 2.2.** *Let $D(S)$ be a disk graph on n sites. We can find a triangle in $D(S)$ in $O(n \log n)$ worst-case time, if it exists.*

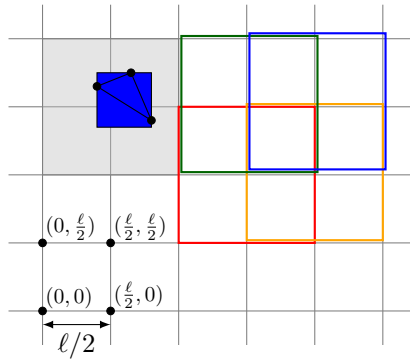
Proof (Sketch). We compute the edges of $D(S)$, using a sweepline approach. If at some point we find more than $3n - 6$ edges, we stop and proceed with the partial graph (which is not plane). If there are at most $3n - 6$ edges, we check for edge crossings, again by a plane sweep. If there is a crossing, we report the resulting triangle in $O(1)$ time. If not, $D(S)$ is plane and we determine if it contains a triangle in $O(n)$ time [12, 17]. ◀

2.2 The Weighted Case

Suppose the edges in $D(S)$ are weighted by their Euclidean lengths. We would like to find a triangle of minimum perimeter, i.e., of minimum total edge length. For this, we solve the decision problem: given $W > 0$, does $D(S)$ contain a triangle with perimeter at most W ? Once a decision algorithm is available, the optimization problem can be solved with Chan’s randomized geometric optimization framework [5].

To decide if $D(S)$ contains a triangle with perimeter at most W , we use a grid with diameter $W/3$. We look for triangles whose vertices lie in a single grid cell, using the algorithm from Section 2.1. If no cell contains such a triangle, then $D(S)$ will be sparse and we will need to check only $O(n)$ further triples. Details follow.

Set $\ell = W/(3\sqrt{2})$. Let G_1 be the grid whose cells are pairwise disjoint, axis-parallel squares with side length ℓ , aligned so that the origin $(0, 0)$ is a vertex of G_1 . The cells of G_1 have diameter $\sqrt{2} \cdot \ell = W/3$, so any triangle whose vertices lie in a single cell has perimeter at most W . We make three additional copies G_2, G_3, G_4 of G_1 , and we shift them by $\ell/2$ in the x -direction, in the y -direction, and in both the x - and y -directions, respectively. In other words, G_2 has $(\ell/2, 0)$ as a vertex, G_3 has $(0, \ell/2)$ as a vertex, and G_4 has $(\ell/2, \ell/2)$ as a vertex, see Figure 1. This ensures that if all edges in a triangle are “short”, the triangle lies in a single grid cell.



■ **Figure 1** The four shifted grids, with a cell from each grid shown in red, orange, green, and blue, respectively. Every square with side length at most $\ell/2$ is wholly contained in a single grid cell.

► **Lemma 2.3.** *Let Δ be a triangle formed by three vertices $a, b, c \in \mathbb{R}^2$ such that each edge of Δ has length at most $\ell/2$. There is a cell $\sigma \in \bigcup_{i=1}^4 G_i$ with $a, b, c \in \sigma$.*

Proof. We can enclose Δ with a square of side length $\ell/2$. This square must be completely contained in a cell of one of the four grids, see Figure 1. ◀

We go through all nonempty grid cells $\sigma \in \bigcup_{i=1}^4 G_i$, and we search for a triangle in the disk graph $D(S \cap \sigma)$ induced by the sites in σ , with Theorem 2.2. Since each site lies in $O(1)$ grid cells, and since we can compute the grid cells for a given site in $O(1)$ time (using the floor function), the total running time is $O(n \log n)$. If a triangle is found, we return YES, since the cells have diameter $W/3$ and thus such a triangle has perimeter at most W . If no triangle is found, Lemma 2.3 implies that any triangle in $D(S)$ has one side of length more than $\ell/2$ and hence at least one vertex with associated radius at least $\ell/4$. We call a site $s \in S$ large if $r_s > \ell/4$. A simple volume argument bounds the number of large sites in a grid cell.

► **Lemma 2.4.** *Let $\sigma \in \bigcup_{i=1}^4 G_i$ be a nonempty grid cell, and suppose that $D(S \cap \sigma)$ does not contain a triangle. Then σ contains at most 18 large sites.*

Proof. Suppose σ contains at least 19 large sites. We cover σ with 3×3 congruent squares of side length $\ell/3$. Then, at least one square τ contains at least $\lceil 19/9 \rceil = 3$ large sites. The associated radius of a large site is more than $\ell/4$ and each square has diameter $(\sqrt{2}/3)\ell < \ell/2$, so the large sites in τ form a triangle in $D(S \cap \sigma)$, a contradiction. ◀

Let $\sigma \in G_i, i \in \{1, \dots, 4\}$, be a grid cell. The neighborhood $N(\sigma)$ of σ is the 5×5 block of cells in G_i centered at σ . Since the diameter of a grid cell is $W/3$, any two sites $u, v \in S$ that form a triangle of perimeter at most W with a site $s \in S \cap \sigma$ must be in $N(\sigma)$. Let $S_\ell \subseteq S$ denote the large sites. At this stage, we know that any triangle in $D(S)$ has at least one vertex in S_ℓ . By Lemma 2.4, for any $\sigma \in \bigcup_{i=1}^4 G_i$, we have $|\bigcup_{\tau \in N(\sigma)} \tau \cap S_\ell| = O(1)$. Thus, to detect a triangle of perimeter at most W with at least two large vertices, we proceed as follows: for each non-empty cell $\sigma \in G_i$, iterate over all large sites s in σ , over all large sites t in $N(\sigma)$, and over all (not necessary large) sites u in $N(\sigma)$. Check whether stu is a triangle of perimeter at most W . If so, return YES. Since the sites in each grid cell are examined $O(1)$ times for $O(1)$ pairs of large sites, the total time is $O(n)$.

It remains to detect triangles of perimeter at most W with exactly one large vertex. We iterate over all grid cells $\sigma \in \bigcup_{i=1}^4 G_i$, and we compute $D(S \cap \sigma)$. Since $D(S \cap \sigma)$ contains no triangle, Lemma 2.1 shows that $D(S \cap \sigma)$ is plane, has $O(|S \cap \sigma|)$ edges and can be

constructed in time $O(|S \cap \sigma| \log |S \cap \sigma|)$. For every edge $st \in D(S \cap \sigma)$ with both endpoints in $S \setminus S_\ell$, we iterate over all large sites u in $N(\sigma)$ and we test whether stu makes a triangle in $D(S)$ with perimeter at most W . If so, we return YES. By Lemma 2.4, this takes $O(|S \cap \sigma|)$ time, so the total running time is $O(n \log n)$. If there is a triangle of perimeter at most W with exactly one vertex in S_ℓ , the edge with both endpoints in $S \setminus S_\ell$ has length at most $\ell/2$ and thus must lie in a single grid cell $\sigma \in \bigcup_{i=1}^4 G_i$. To summarize:

► **Lemma 2.5.** *Let $D(S)$ be a disk graph on n sites, and let $W > 0$. We can decide in $O(n \log n)$ worst-case time whether $D(S)$ contains a triangle of perimeter at most W .*

Now, Chan's framework [5] gives a randomized optimization algorithm with no additional overhead. The details can be found in the full version.

► **Theorem 2.6.** *Let $D(S)$ be a weighted disk graph on n sites. We can compute a shortest triangle in $D(S)$ in $O(n \log n)$ expected time, if one exists.*

3 Computing the Girth of a Disk Graph

We extend the results from Section 2 to the girth. The unweighted case is easy: if $D(S)$ is not plane, the girth is 3, by Lemma 2.1. If $D(S)$ is plane, we use the algorithm for planar graphs [7]. The weighted case is harder. If $D(S)$ is plane, we use the algorithm for planar graphs [15]. If not, Theorem 2.6 gives a shortest triangle Δ in $D(S)$. However, there could be cycles with at least four edges that are shorter than Δ . To address this, we use Δ to split $D(S)$ into sparse pieces where a shortest cycle can be found efficiently.

3.1 The Unweighted Case

Chang and Lu [7, Theorem 1.1] showed how to find the girth of an unweighted planar graph with n vertices in $O(n)$ time. Hence, we obtain a simple extension of Theorem 2.2.

► **Theorem 3.1.** *Let $D(S)$ be a disk graph for a set S of n sites. We can compute the unweighted girth of $D(S)$ in $O(n \log n)$ worst-case time.*

Proof. We proceed as in Theorem 2.2. If $D(S)$ is not plane, the girth is 3. If $D(S)$ is plane, we apply the algorithm of Chang and Lu [7, Theorem 1.1] to an explicit representation of $D(S)$. ◀

3.2 The Weighted Case

We describe how to find the shortest cycle through a given vertex in a weighted graph with certain properties. This is then used to compute the weighted girth of a disk graph.

Let G be a graph with nonnegative edge weights so that all shortest paths and cycles in G have pairwise distinct lengths and so that for all edges uv , the shortest path from u to v is the edge uv . We present a deterministic algorithm that, given G and a vertex s , computes the shortest cycle in G containing s , if it exists. A simple randomized algorithm can also be found in Yuster [23, Section 2]. The next lemma states a structural property of the shortest cycle through s . It resembles Lemma 1 of Roditty and V. Williams [19] that deals with an overall shortest cycle in G .² For details on the proof see the full version.

² Even though this seems to be a simple fact, we could not locate a previous reference for this.

► **Lemma 3.2.** *The shortest cycle in G that contains s consists of two paths in the shortest path tree of s , and one additional edge.*

► **Theorem 3.3.** *Let $G = (V, E)$ be a weighted graph with n vertices and m edges that has the properties given at the beginning of this section. Let $s \in V$. We can compute the shortest cycle in G that contains s in $O(n \log n + m)$ time, if it exists.*

Proof (Sketch). We find the shortest path tree T for s , and we identify the edges that close a cycle in T containing s . We check all candidate cycles to find the shortest. Correctness follows from Lemma 3.2. The running time for finding the shortest path tree dominates the rest of the algorithm. ◀

Let $D(S)$ be a weighted disk graph on n sites. A careful combination of the tools developed so far gives an algorithm for the weighted girth of $D(S)$.

► **Theorem 3.4.** *Given a weighted disk graph $D(S)$ on n sites, we can compute the weighted girth of $D(S)$ in $O(n \log n)$ expected time.*

Proof (Sketch). We find the shortest triangle in $D(S)$ in $O(n \log n)$ expected time, if it exists (Theorem 2.6). If $D(S)$ has no triangle, it is plane by Lemma 2.1, and we construct $D(S)$ in $O(n \log n)$ time. We find the girth of $D(S)$ using the algorithm of Łącki and Sankowski [15, Section 5], in $O(n \log \log n)$ time. If $D(S)$ has a triangle, let W be the perimeter of the shortest triangle in $D(S)$. Then, W is an upper bound for the girth of $D(S)$. We set $\ell = W/(3\sqrt{2})$, and we call a site $s \in S$ *large*, if $r_s \geq \ell/4$, and we write $S_\ell \subseteq S$ for the set of large sites. Let G be the grid with side length ℓ and the origin $(0, 0)$ as a vertex.

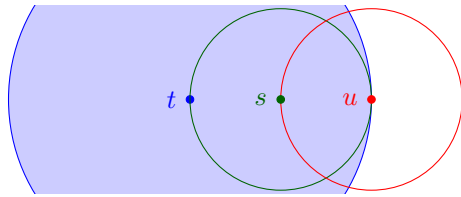
We must check whether $D(S)$ contains a cycle with more than three vertices and length less than W . By our choice of ℓ , the graph $D(S \setminus S_\ell)$ induced by $S \setminus S_\ell$ is plane. Thus, we can find a cycle in $D(S \setminus S_\ell)$ with the algorithm of Łącki and Sankowski [15, Section 5], in $O(n \log \log n)$ time. It remains to test cycles with at least one large site. The choice of ℓ implies that each cycle of length at most W is contained in a grid neighborhood of constant size. Since there are $O(1)$ large sites in each neighborhood, the induced graph in each neighborhood has linear size. We check the remaining cycles by applying Theorem 3.3 to all large sites in each neighborhood. ◀

4 Finding a Triangle in a Transmission Graph

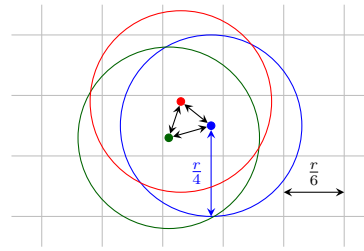
Given a transmission graph $T(S)$ on n sites, we want to decide if $T(S)$ contains a directed triangle. We first describe an inefficient algorithm for this problem, and then we will explain how to implement it in $O(n \log n)$ expected time.

The algorithm iterates over each directed edge $e = st$ with $r_t \geq r_s$, and it performs two tests: first, for each directed edge tu with $r_u \geq r_t/2$, it checks if us is an edge in $T(S)$, i.e., if $s \in D_u$. If so, the algorithm reports the triangle stu . Second, the algorithm tests if there is a site u such that $r_u \in [r_s, r_t/2)$ and such that us is an edge in $T(S)$, i.e., such that $s \in D_u$. If such a u exists, it reports the triangle stu . If both tests fail for each edge e , the algorithm reports that $T(S)$ contains no triangle. The next lemma shows that the algorithm is correct.

► **Lemma 4.1.** *A triple stu reported by the algorithm is a triangle in $T(S)$. Furthermore, if $T(S)$ contains a triangle, the algorithm will find one.*



(a) We do not need to check $u \in D_t$.



(b) Three disks with radius at least $r/4$ in the same grid cell form a clique.

Proof (Sketch). It is easy to see that the algorithm finds a triangle if one exists. The algorithm is also sound: a triple reported by the first test is a triangle by construction, and for the second test, Figure 2a shows that $u \in D_t$, so tu is an edge of $T(S)$. ◀

There are several challenges for making the algorithm efficient. First of all, there might be many edges st with $r_t \geq r_s$. However, the following lemma shows that if there are $\omega(n)$ such edges, the transmission graph $T(S)$ must contain a triangle.

► **Lemma 4.2.** *There is an absolute constant α so that for any $r > 0$, if there is an $r \times r$ square σ that contains more than α sites $s \in S$ with $r_s \geq r/4$, then $T(S)$ has a directed triangle.*

Proof. We cover σ with a 6×6 grid of side length $r/6$; see Figure 2b. There are 36 grid cells. For every $s \in S \cap \sigma$ with $r_s \geq r/4$, the disk D_s completely covers the grid cell containing s . If σ contains more than $\alpha = 72$ sites s with $r_s \geq r/4$, then one grid cell contains at least three such sites. These sites form a directed triangle in $T(S)$. ◀

Thus, to implement the algorithm, we must solve two range searching problems.

(R1) EITHER determine that for every site $s \in S$, there are at most α outgoing edges st with $r_t \geq r_s/2$ and report all these edges; OR find a square σ of side length $r > 0$ that contains more than α sites $s \in S$ with $r_s \geq r/4$.

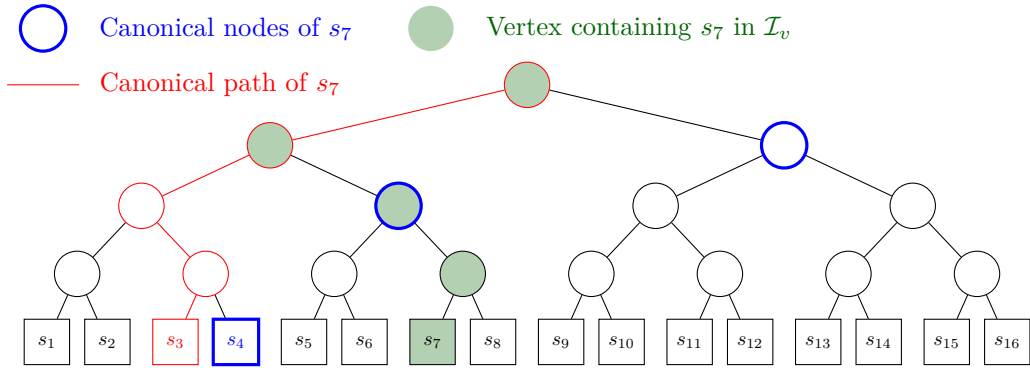
(R2) Given $O(n)$ query triples (s, r_1, r_2) with $s \in S$ and $0 < r_1 < r_2$, find a site $u \in S$ such that there is a query triple (s, r_1, r_2) with $u \neq s$, $r_u \in [r_1, r_2)$, and $s \in D_u$; or report that no such site exists.

The query (R1) indeed always has a valid outcome: suppose there is a site $s \in S$ with more than α outgoing edges st with $r_t \geq r_s/2$. Then, all the endpoints t lie in D_s , so the square σ centered at s with side length $r = 2r_s$ contains more than α sites with associated radius at least $r/4$. The next theorem shows that we can detect a triangle in $T(S)$ with linear overhead in addition to the time needed for answering (R1) and (R2). The proof is in the full version.

► **Theorem 4.3.** *If (R1) and (R2) can be solved in time $R(n)$ for input size n , we can find a directed triangle in a transmission graph $T(S)$ on n sites in time $R(n) + O(n)$, if it exists.*

In the next section, we will implement (R1) and (R2) in $O(n \log n)$ expected time.

► **Theorem 4.4.** *Let $T(S)$ be a transmission graph on n sites. We can find a directed triangle in $T(S)$ in expected time $O(n \log n)$, if it exists.*



■ **Figure 3** Example is for a query of type (R1), assuming that $r_{s_3} < r_{s_7}/2 \leq r_{s_4}$.

5 Batched Range Searching

The range queries must handle subsets of sites whose associated radii lie in certain intervals: a query s in (R1) concerns sites $t \in S$ such that $r_t \geq r_s/2$; and a query (s, r_1, r_2) in (R2) concerns sites t such that $r_t \in [r_1, r_2)$. Using a standard approach [2, 20], we subdivide each such *query interval* into $O(\log n)$ pieces from a set of *canonical intervals*. For this, we build a balanced binary tree B whose leaves are the sites of S , sorted by increasing associated radius. For each vertex $v \in B$, let the *canonical interval* \mathcal{I}_v be the sorted list of sites in the subtree rooted at v . There are $O(n)$ canonical intervals.

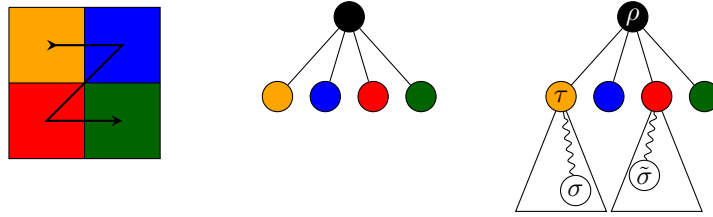
Next, we define *canonical paths* and *canonical nodes*. For a radius $r > 0$, the (proper) *predecessor* of r is the site $s \in S$ with the largest radius $r_s \leq r$ ($r_s < r$). The (proper) *successor* of r is defined analogously. For a query s in (R1), we consider the path π in B from the root to the leaf with the proper predecessor t of $r_s/2$. If t does not exist (i.e., if $r_t \geq r_s/2$, for all $t \in S$), we let π be the left spine of B . We call π the *canonical path* for s . The *canonical nodes* for s are the right children of the nodes in π that are not in π themselves, plus possibly the last node of π , if $r_t \geq r_s/2$, for all $t \in S$, see Figure 3.

For a query (s, r_1, r_2) in (R2), we consider the path π_1 in B from the root to the leaf with the proper predecessor t_1 of r_1 and the path π_2 in B from the root to the leaf for the successor t_2 of r_2 . Again, if t_1 does not exist, we take π_1 as the left spine of B , and if t_2 does not exist, we take π_2 as the right spine of B . Then, π_1 and π_2 are the *canonical paths* for (s, r_1, r_2) . The *canonical nodes* for (s, r_1, r_2) are defined as follows: for each vertex v in $\pi_1 \setminus \pi_2$, we take the right child of v if it is not in π_1 , and for each v in $\pi_2 \setminus \pi_1$, we take the left child of v if it is not in π_1 . Furthermore, we take the last node of π_1 if t_1 does not exist, and the last node of π_2 if t_2 does not exist. A standard argument bounds the number and total size of the canonical intervals, see the full version for the proof.

► **Lemma 5.1.** *The total size of the canonical intervals is $O(n \log n)$. The tree B and the canonical intervals can be built in $O(n \log n)$ time. For any query q in (R1) or (R2), there are $O(\log n)$ canonical nodes, and they can be found in $O(\log n)$ time. The canonical intervals for the canonical nodes of q constitute a partition of the query interval for q .*

5.1 Queries of Type (R1)

We build a compressed quadtree on S , and we perform the range searches the compressed quadtree. It is possible to compute a compressed quadtree for each canonical interval without logarithmic overhead. Since Lemma 4.2 gives us plenty of freedom in choosing the squares



■ **Figure 4** Z-Order. On the very right we have $\sigma \leq_Z \tau \leq_Z \tilde{\sigma}$.

for our range queries, we take squares from the grid that underlies the quadtree. This allows us to reduce the range searching problem to predecessor search in a linear list, a task that can be accomplished by one top-down traversal of B . Details follow.

Hierarchical grids, Z-order, compressed quadtrees. We translate and scale S (and the associated radii), so that S lies in the interior of the unit square $U = [0, 1]^2$ and so that all radii are at most $\sqrt{2}$. We define a sequence of hierarchical grids that subdivide U . The grid G_0 consists of the single cell U . The grid G_i , $i \geq 1$, consists of the 2^{2i} square cells with side length 2^{-i} and pairwise disjoint interiors that cover U . The hierarchical grids induce an infinite four-regular tree \mathcal{T} : the vertices are the cells of $\mathcal{G} = \bigcup_{i=0}^{\infty} G_i$. The unit square U is the root, and for $i = 1, \dots$, a cell σ in G_i is the child of the cell in G_{i-1} that contains it. We make no explicit distinction between a vertex of \mathcal{T} and its corresponding cell.

The *Z-order* \leq_Z is a total order on the cells of \mathcal{G} ; see [4] for more details. Let $\sigma, \tau \in \mathcal{G}$. If $\sigma \subseteq \tau$, then $\sigma \leq_Z \tau$; and if $\tau \subseteq \sigma$, then $\tau \leq_Z \sigma$. If σ and τ are unrelated in \mathcal{T} , let ρ be the lowest common ancestor of σ and τ in \mathcal{T} , and let σ' and τ' be the children of ρ with $\sigma \subseteq \sigma'$ and $\tau \subseteq \tau'$. We set $\sigma \leq_Z \tau$ if σ' is before τ' in the order shown in Figure 4; and $\tau \leq_Z \sigma$, otherwise. The next lemma shows that given $\sigma, \tau \in \mathcal{G}$, we can decide if $\sigma \leq_Z \tau$ in constant time.

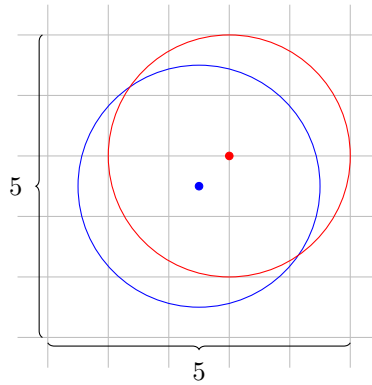
► **Lemma 5.2** (Chapter 2 in Har-Peled [11]). *Suppose the floor function and the first differing bit in the binary representations of two given real numbers can be computed in $O(1)$ time. Then, we can decide in $O(1)$ time for two given cells $\sigma, \tau \in \mathcal{G}$ whether $\sigma \leq_Z \tau$ or $\tau \leq_Z \sigma$.*

For a site $s \in S$, let σ_s be the largest cell in \mathcal{G} that contains only s . The *quadtree* for S is the smallest connected subtree of \mathcal{T} that contains the root U and all cells σ_s , for $s \in S$. The *compressed quadtree* \mathcal{C} for S is obtained from the quadtree by contracting any maximal path of vertices with only one child into a single edge. Vertices that were at the top of such a path are now called *compressed vertices*. The compressed quadtree for S has $O(n)$ vertices, and it can be constructed in $O(n \log n)$ time (see, e.g., [3, Appendix A] and [11]).

The *linearized compressed quadtree* \mathcal{L} for S is the sorted sequence of cells obtained by listing the nodes of \mathcal{C} according to a postorder traversal, where the children of a node $\sigma \in \mathcal{C}$ are visited according to the Z-order from Figure 4. The cells in \mathcal{L} appear in increasing Z-order, and range searching for a given cell $\sigma \in \mathcal{G}$ reduces to a simple predecessor search in \mathcal{L} , as is made explicit in the following lemma.

► **Lemma 5.3.** *Let σ be a cell of \mathcal{G} , and let \mathcal{L} be the linearized compressed quadtree on S . Let $\tau = \max_Z\{\rho \in \mathcal{L} \mid \rho \leq_Z \sigma\}$ be the Z-predecessor of σ in \mathcal{L} ($\tau = \emptyset$, if the predecessor does not exist). Then, if $\sigma \cap \tau = \emptyset$, then also $\sigma \cap S = \emptyset$, and if $\sigma \cap \tau \neq \emptyset$, then $\sigma \cap S = \tau \cap S$.*

Proof. Let \mathcal{C} be the compressed quadtree on S , and let $\mathcal{C}_\sigma = \{\tau \in \mathcal{C} \mid \tau \subseteq \sigma\}$ be the cells in \mathcal{C} that are contained in σ . If \mathcal{C}_σ is non-empty, then \mathcal{C}_σ is a connected subtree of \mathcal{C} . Let τ be the root of this subtree. Then, $\tau = \max_Z\{\rho \in \mathcal{C}_\sigma\}$, and $\tau \leq_Z \sigma$. Furthermore, all other cells



■ **Figure 5** The neighborhood of a site has constant size.

in $\mathcal{C} \setminus \mathcal{C}_\sigma$ are either smaller than all cells in \mathcal{C}_σ or larger than σ . Thus, τ is the Z -predecessor of σ in \mathcal{L} , and $\sigma \cap S = \tau \cap S \neq \emptyset$. Otherwise, if $\mathcal{C}_\sigma = \emptyset$, the Z -predecessor of σ in \mathcal{L} either does not exist or is disjoint from σ . Thus, in this case, we have $\emptyset = \sigma \cap \tau = \sigma \cap S$. ◀

The search algorithm. For a site $s \in S$, we define the *neighborhood* $N(s)$ of s as all cells in \mathcal{G} with side length $2^{\lceil \log_2 r_s \rceil}$ that intersect D_s . The neighborhood will be used to approximate D_s for the range search in the quadtrees.

► **Lemma 5.4.** *There is a constant β such that $|N(s)| \leq \beta$ for all $s \in S$.*

Proof. We have $r_s/2 < 2^{\lceil \log_2 r_s \rceil}$, and a 5×5 grid with cells of side length $r_s/2$ covers D_s , no matter where s lies; see Figure 5. Thus, the lemma holds with $\beta = 25$. ◀

We now show that a linearized compressed quadtree for each canonical interval can be found without logarithmic overhead.

► **Lemma 5.5.** *We can compute for each $v \in B$ the linearized quadtree \mathcal{L}_v for the sites in \mathcal{I}_v in $O(n \log n)$ time.*

Proof (Sketch). We traverse B and build the compressed quadtree \mathcal{C}_v for \mathcal{I}_v , for each $v \in B$. For the root, this takes $O(n \log n)$ time. The compressed quadtree \mathcal{C}_w for a child w of a node v can be found by traversing \mathcal{C}_v in $O(|\mathcal{C}_v|)$ time. Then, we compute the linearized trees \mathcal{L}_v by a postorder traversal of each \mathcal{C}_v . ◀

Using the linearized compressed quadtrees, the range searching problem can be solved by a batched predecessor search, using a single traversal of B .

► **Lemma 5.6.** *The range searching problem (R1) can be solved in $O(n \log n)$ time.*

Proof (Sketch). We apply Lemma 5.5 to find the linearized quadtrees for B . Let $\mathcal{Q}' = \bigcup_{s \in S} \{(\sigma, s) \mid \sigma \in N(s)\}$. We call \mathcal{Q}' the set of *split queries*. They approximate the disks D_s by cells from the hierarchical grid. By Lemma 5.4, $|\mathcal{Q}'| = O(n)$. We now want to perform range queries for all the cells in the split queries. For this, we first sort the elements of \mathcal{Q}' in the Z -order of their first components, in $O(n \log n)$ time. Now we store all split queries (σ, s) with all nodes $v \in B$ such that v is a canonical node of s . This can be done by one traversal over B . We call the resulting lists \mathcal{Q}''_v , for $v \in B$.

We iterate over all $v \in B$, and we merge the lists \mathcal{Q}'_v with the lists \mathcal{L}_v , in Z -order. This takes $O(\sum_{v \in B} |\mathcal{L}_v| + |\mathcal{Q}'_v|) = O(n \log n)$ time. By Lemma 5.3, we obtain for each $(\sigma, s) \in \mathcal{Q}''_v$ a cell $\tau_v^{\sigma, s}$. If $\sigma \cap \tau_v^{\sigma, s} \neq \emptyset$, we know that $\sigma \cap \mathcal{I}_v = \tau_v^{\sigma, s} \cap \mathcal{I}_v$. Since these sites are all from \mathcal{I}_v they all have radius at least $r_s/2$. We can find all these sites in $O(k)$ time, where k is the output size. If $k > \alpha$, we stop and report σ as a square with many sites of large radius.³

Otherwise, we use the sites in σ to accumulate the sites for the query disk D_s . This we can do by considering all canonical nodes of s and for each cell σ iterate over the sites contained in σ . In each such cell there are at most α sites. For each site $t \in \sigma$ we can check if $t \in D_s$. If we find a query disk D_s with more than α sites of large radius, we stop and report its enclosing square with many sites of large radius.⁴ Otherwise, for each $s \in S$, we have found the at most α sites of radius at least $r_s/2$ in D_s . The whole algorithm takes $O(n \log n)$ time. ◀

5.2 Queries of Type (R2)

We use the tree structure of the canonical intervals (i) to construct quickly the search structures for each canonical interval; and (ii) to solve all queries for a canonical interval in one batch. We exploit the connection between planar disks and three-dimensional polytopes. Let $U = \{(x, y, z) \mid x^2 + y^2 = z\}$ be the three-dimensional unit paraboloid. For a site $s \in S$, the lifted site \hat{s} is the vertical projection of s onto U . Each disk D_s is transformed into an upper halfspace \widehat{D}_s , so that the projection of $\widehat{D}_s \cap U$ onto the xy -plane is the set $\mathbb{R}^2 \setminus D_s$;⁵ see Figure 6. The union of a set of disks in \mathbb{R}^2 corresponds to the intersection of the lifted upper halfspaces in \mathbb{R}^3 .

▶ **Lemma 5.7.** *The range searching problem (R2) can be solved in $O(n \log n)$ expected time.*

Proof (Sketch). For each $v \in B$, we construct the intersection \mathcal{E}_v of the \widehat{D}_s , $s \in \mathcal{I}_v$. We can find all the polyhedra \mathcal{E}_v , for $v \in B$, in overall $O(n \log n)$ time, by a single traversal of B . This traversal goes bottom up and uses that we can find the intersection of two convex polyhedra in $O(n)$ time [6]. For batched query processing, we need for each $v \in B$ the convex hull of the lifted query sites that have v as a canonical node. These convex hulls can be computed top-down by splitting the current convex hulls in each node in $O(n \log n)$ overall time, using linear time per node [8]. We call these hulls \widehat{Q}_v .

To answer all queries, we use the polyhedra \widehat{Q}_v and \mathcal{E}_v . For each node $v \in B$, we compute the lifted sites in \widehat{Q}_v that are not inside of \mathcal{E}_v . These sites correspond to the vertices of \widehat{Q}_v that are not vertices of $\widehat{Q}_v \cap \mathcal{E}_v$. These intersections can be found in overall $O(n \log n)$ time, again by using the fact that we can intersect two polyhedra in $O(n)$ time. If for any such intersection $\widehat{Q}_v \cap \mathcal{E}_v$, there is a lifted site $\hat{s} \in \widehat{Q}_v$ that is not a vertex of $\widehat{Q}_v \cap \mathcal{E}_v$, we report s as the result of the range search. Otherwise, we report our range search to be unsuccessful. ◀

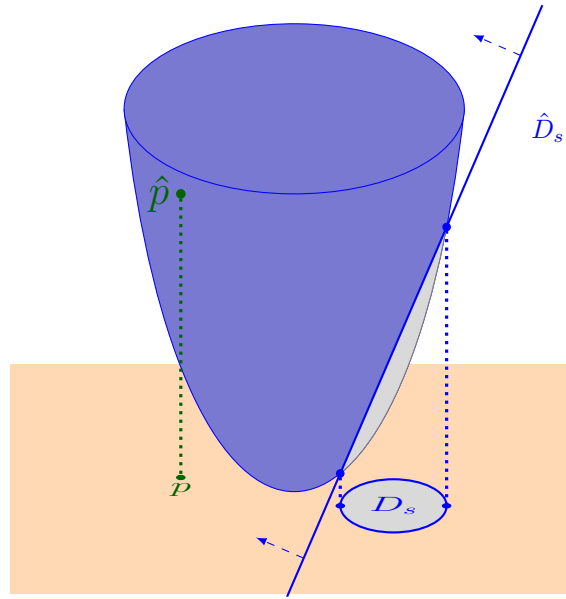
6 Finding the Shortest Triangle in a Transmission Graph

We extend Theorem 4.4 to find the shortest triangle in $T(S)$. As in Section 2.2, we solve the decision problem: given $W > 0$, does $T(S)$ have a directed triangle of perimeter at most W ? We set $\ell = W/\sqrt{27}$, and call a site $s \in S$ *large* if $r_s > \ell$. We let $S_\ell \subseteq S$ be the set of all large sites.

³ Note that here the radii are $\geq r_s/2$ inside of the cells σ . This might be larger than the value $2^{\lceil \log_2 r_s \rceil} / 4$ needed by (R1). But still, if there are more than α sites in σ , we still have a triangle in a square. Otherwise we will later determine that each disk contains few sites of radius at least $r_s/2$.

⁴ $r = 2r_s$ is the side length of the enclosing square, the radii are at least $r/4$ as desired.

⁵ This halfspace is bounded by the plane $z = 2x_s x - x_s^2 + 2y_s y - y_s^2 + r_s^2$, where $s = (x_s, y_s)$.



■ **Figure 6** Lifting disks and points. For \hat{D} only the bounding plane is shown.

► **Lemma 6.1.** *We can find a triangle in $T(S \setminus S_\ell)$ of perimeter at most W in $O(n \log n)$ time, if it exists.*

Proof. Any triangle in $T(S \setminus S_\ell)$ has perimeter at most W : consider a directed triangle stu in $T(S \setminus S_\ell)$ with $r_s \geq \max\{r_t, r_u\}$. Then we have $t, u \in D_s$, so the triangle stu lies in D_s . Elementary calculus shows that a triangle of maximum perimeter in D_s must be equilateral with its vertices on ∂D_s , so any triangle contained in D_s has perimeter at most $3 \cdot \sqrt{3} \cdot r_s \leq \sqrt{27} \cdot \ell = W$. We can find a triangle in T' in $O(n \log n)$ time by Theorem 4.4. ◀

It remains to check for triangles of perimeter at most W with at least one large vertex. Some such triangles have to be considered individually, while the others can be handled efficiently in batch mode. The following lemma shows that we may assume that there are few edges from $S \setminus S_\ell$ to S_ℓ .

► **Lemma 6.2.** *If $T(S)$ does not have a triangle of perimeter at most W , every site in S_ℓ has at most six incoming edges from $S \setminus S_\ell$. Furthermore, in $O(n \log n)$ time, we can either find a triangle of perimeter at most W in $T(S)$ or determine for each site in S_ℓ all incoming edges from $S \setminus S_\ell$.*

Proof (Sketch). By a suitable variant of (R1), we either find a triangle of perimeter at most W , or we restrict the overall number of edges from $S \setminus S_\ell$ to S_ℓ to $O(n)$ in $O(n \log n)$ time. To bound the indegree of the large sites, we observe that if a large site has 7 incoming edges from $S \setminus S_\ell$, then there is a triangle in $T(S)$ of perimeter at most W . ◀

Next, we want to limit the number of relevant edges between large sites. For this, we subdivide the plane with a grid G of side length $\ell/\sqrt{2}$. Then, we have the following:

► **Lemma 6.3.** *A triangle contained in a cell $\sigma \in G$ has perimeter at most W . If there is no triangle in σ , then σ contains $O(1)$ large sites. We can check for such triangles in $O(n \log n)$ overall expected time.*

Proof. The maximum perimeter of a triangle contained in σ is $(1 + \sqrt{2})\ell < W$. Furthermore, if there are at least three large sites in σ , these large sites form a triangle, since the disk of a large site covers σ . By applying Theorem 4.4 to the induced subgraph in each cell of G , we can find such a triangle in $O(n \log n)$ total expected time. ◀

We define the neighborhood $N(\sigma)$ of a cell $\sigma \in G$ as the 5×5 block of cells centered at σ . Let t be a site and σ the cell containing t , then the neighborhood $N(t)$ of t are all sites contained in $N(\sigma)$. Since the side length of a grid cell is $W/3\sqrt{6}$, each triangle of perimeter at most W is completely contained in the neighborhood of some cell.

► **Lemma 6.4.** *We can check the remaining triangles in $O(n)$ overall time.*

Proof. Consider a remaining triangle sut with $r_t \geq \max\{r_u, r_s\}$. Then, $t \in S_\ell$, and s, t, u all lie in $N(t)$. By Lemma 6.3, there are $O(1)$ large candidates for u in $N(t)$, and by Lemma 6.2, there are $O(1)$ small candidates for u . Having fixed a t and a possible candidate u , we iterate over all $s \in N(t)$ and check if s, u , and t form a triangle with weight at most W . Every site s is contained in $O(1)$ grid neighborhoods, and since there are $O(1)$ candidate pairs in each grid neighborhood, s participates in $O(1)$ explicit checks. The result follows. ◀

The following theorem summarizes the considerations in this section.

► **Theorem 6.5.** *It takes $O(n \log n)$ expected time to find the shortest triangle in a transmission graph.*

Proof. We already saw that there is an $O(n \log n)$ time decision algorithm for the problem. As in Theorem 2.6, the result follows from an application of Chan's randomized optimization technique [5]. ◀

7 Conclusion

Once again, disk graphs and transmission graphs prove to be a simple yet powerful graph model where difficult algorithmic problems admit faster solutions. It would be interesting to find a *deterministic* $O(n \log n)$ time algorithm for finding a shortest triangle in a disk graph. Currently, we are working on extending our results to the girth problem in transmission graphs; can we find an equally simple and efficient algorithm as for disk graphs?

References


- 1 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and Counting Given Length Cycles. *Algorithmica*, 17(3):209–223, 1997.
- 2 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, third edition, 2008.
- 3 Kevin Buchin, Maarten Löffler, Pat Morin, and Wolfgang Mulzer. Preprocessing Imprecise Points for Delaunay Triangulation: Simplified and Extended. *Algorithmica*, 61(3):674–693, 2011.
- 4 Kevin Buchin and Wolfgang Mulzer. Delaunay triangulations in $O(\text{sort}(n))$ time and more. *J. ACM*, 58(2):6:1–6:27, 2011.
- 5 Timothy M. Chan. Geometric Applications of a Randomized Optimization Technique. *Discrete Comput. Geom.*, 22(4):547–567, 1999.
- 6 Timothy M. Chan. A Simpler Linear-Time Algorithm for Intersecting Two Convex Polyhedra in Three Dimensions. *Discrete Comput. Geom.*, 56(4):860–865, December 2016.
- 7 Hsien-Chih Chang and Hsueh-I Lu. Computing the Girth of a Planar Graph in Linear Time. *SIAM J. Comput.*, 42(3):1077–1094, 2013.

- 8 Bernard Chazelle and Wolfgang Mulzer. Computing Hereditary Convex Structures. *Discrete Comput. Geom.*, 45(4):796–823, 2011.
- 9 William S. Evans, Mereke van Garderen, Maarten Löffler, and Valentin Polishchuk. Recognizing a DOG is Hard, But Not When It is Thin and Unit. In *Proc. 8th FUN*, pages 16:1–16:12, 2016.
- 10 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. 39th Internat. Symp. Symbolic and Algebraic Comput. (ISSAC)*, pages 296–303, 2014.
- 11 Sarel Har-Peled. *Geometric approximation algorithms*. American Mathematical Society, 2008.
- 12 Alon Itai and Michael Rodeh. Finding a Minimum Circuit in a Graph. *SIAM J. Comput.*, 7(4):413–423, 1978.
- 13 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Spanners and Reachability Oracles for Directed Transmission Graphs. In *Proc. 31st Int. Sympos. Comput. Geom. (SoCG)*, pages 156–170, 2015.
- 14 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Spanners for Directed Transmission Graphs. *SIAM J. Comput.*, 47(4):1585–1609, 2018.
- 15 Jakub Łącki and Piotr Sankowski. Min-cuts and shortest cycles in planar graphs in $O(n \log \log n)$ time. In *Proc. 19th Annu. European Sympos. Algorithms (ESA)*, pages 155–166, 2011.
- 16 Shay Mozes, Kirill Nikolaev, Yahav Nussbaum, and Oren Weimann. Minimum Cut of Directed Planar Graphs in $O(n \log \log n)$ Time. In *Proc. 29th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 477–494, 2018.
- 17 Christos H. Papadimitriou and Mihalis Yannakakis. The Clique Problem for Planar Graphs. *Inform. Process. Lett.*, 13(4/5):131–133, 1981.
- 18 Valentin Polishchuk. Personal communication, 2017.
- 19 Liam Roditty and Virginia Vassilevska Williams. Minimum Weight Cycles and Triangles: Equivalences and Algorithms. In *Proc. 52nd Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 180–189, 2011.
- 20 Dan E. Willard and George S. Lueker. Adding Range Restriction Capability to Dynamic Data Structures. *J. ACM*, 32(3):597–617, 1985.
- 21 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic Equivalences Between Path, Matrix, and Triangle Problems. *J. ACM*, 65(5):27:1–27:38, 2018.
- 22 Huacheng Yu. An Improved Combinatorial Algorithm for Boolean Matrix Multiplication. In *Proc. 42nd Internat. Colloq. Automata Lang. Program. (ICALP)*, pages 1094–1105, 2015.
- 23 Raphael Yuster. A shortest cycle for each vertex of a graph. *Inform. Process. Lett.*, 111(21-22):1057–1061, 2011.

Reliable Hubs for Partially-Dynamic All-Pairs Shortest Paths in Directed Graphs

Adam Karczmarz 

Institute of Informatics, University of Warsaw, Poland
a.karczmarz@mimuw.edu.pl

Jakub Łącki 

Google Research, New York, USA
jlacki@google.com

Abstract

We give new partially-dynamic algorithms for the all-pairs shortest paths problem in weighted directed graphs. Most importantly, we give a new *deterministic* incremental algorithm for the problem that handles updates in $\tilde{O}(mn^{4/3} \log W/\epsilon)$ total time (where the edge weights are from $[1, W]$) and explicitly maintains a $(1 + \epsilon)$ -approximate distance matrix. For a fixed $\epsilon > 0$, this is the first deterministic partially dynamic algorithm for all-pairs shortest paths in directed graphs, whose update time is $o(n^2)$ regardless of the number of edges. Furthermore, we also show how to improve the state-of-the-art partially dynamic *randomized* algorithms for all-pairs shortest paths [Baswana et al. STOC'02, Bernstein STOC'13] from Monte Carlo randomized to Las Vegas randomized without increasing the running time bounds (with respect to the $\tilde{O}(\cdot)$ notation).

Our results are obtained by giving new algorithms for the problem of dynamically maintaining *hubs*, that is a set of $\tilde{O}(n/d)$ vertices which hit a shortest path between each pair of vertices, provided it has hop-length $\Omega(d)$. We give new subquadratic deterministic and Las Vegas algorithms for maintenance of hubs under either edge insertions or deletions.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases shortest paths, dynamic, incremental, decremental, directed graphs, hubs

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.65

Related Version A full version of the paper is available at <https://arxiv.org/abs/1907.02266>.

Funding *Adam Karczmarz*: Supported by ERC Consolidator Grant 772346 TUGBOAT and the Polish National Science Centre 2018/29/N/ST6/00757 grant.

1 Introduction

The sampling scheme of Ullman and Yannakakis [27] is a fundamental tool in designing dynamic algorithms for maintaining shortest path distances. Roughly speaking, the main idea is that if each vertex of the graph is sampled independently with probability $\Omega(\frac{d \ln n}{n})$, then with high probability¹ the set of the sampled vertices has the following property. If the shortest path between some vertices u and v contains more than d edges, then this shortest path contains a sampled vertex². We call each set having this property a set of *hubs*³ of that graph.

¹ We say that a probabilistic statement holds *with high (low) probability*, abbreviated w.h.p., if it holds with probability at least $1 - n^{-\beta}$ (at most $n^{-\beta}$, resp.), where β is a constant that can be fixed arbitrarily.

² For simplicity, in the introduction we assume that the shortest paths are unique.

³ Zwick [29] uses the name *bridging set* for an analogous concept. Some works also use the term *hitting set*, but hitting set is a more general notion, which in our paper is used in multiple different contexts.



The fact that one can easily obtain a set of hubs by random sampling is particularly useful for dynamic graph algorithms, since, by tuning constants in the sampling probability, one can assure that the set of hubs remains valid at each step (with high probability), while the graph is undergoing edge insertions and deletions, assuming the total number of updates is polynomial. This property has been successfully exploited to give a number of dynamic graph algorithms, e.g. [2, 4, 5, 9, 12, 11, 13, 20, 21, 22]. At the same time, the sampling approach also suffers from two drawbacks. First, it yields Monte Carlo algorithms, which with some nonzero probability can return incorrect answers. Second, it relies on the *oblivious adversary* assumption, that is, it requires that the updates to the graph are independent of the randomness used for sampling hubs. This becomes a substantial issue for problems where the answer to a query is not unique, e.g., for maintaining $(1 + \epsilon)$ -approximate distances or maintaining the shortest paths themselves (i.e. not just their lengths). In a typical case, the choice of the specific answer to a query depends on the randomness used for vertex sampling, which in turn means that in each answer to a query the data structure is revealing its randomness. Hence, if the following updates to the data structure depend on the specific values returned by the previous queries, the oblivious adversary assumption is not met.

In this paper we attempt to address both these issues. We study the dynamic maintenance of *reliable* hubs, that is we show how to maintain hubs using an algorithm that does not err, even with small probability. In addition, in the incremental setting we give an algorithm that maintains hubs deterministically. While the algorithms are relatively straightforward for unweighted graphs, making them also work in the weighted setting is a major challenge, which we manage to overcome. We then show how to take advantage of our results on reliable hubs to obtain improved algorithms for the problem of maintaining all-pairs shortest paths in directed graphs. In particular, we give a faster deterministic incremental algorithm and show how to improve the state-of-the-art decremental algorithms from Monte Carlo to Las Vegas.

1.1 Our Contribution

We study the problem of maintaining *reliable* hub sets in the partially dynamic setting. For the description, let us first assume the case when the graph is unweighted. Our first observation is that one can deterministically maintain the set of hubs H_d under edge insertions in $\tilde{O}(nmd)$ total time. To that end, we observe that after an edge uw is inserted, we may ensure the set of hubs H_d is valid by extending it with both u and w . This increases the size of H_d , and hence we have to periodically discard all the hubs and recompute them from scratch.

The deterministic computation of hubs has been studied before. For unweighted digraphs, King [19] showed how to compute a hub set H_d of size $\tilde{O}\left(\frac{n}{d}\right)$ in $\tilde{O}(n^2)$ time. The algorithm, given shortest path trees up to depth d from all vertices $v \in V$, computes a *blocker-set* [19] of these trees. (A blocker-set S of a rooted tree is a set such that, for each path from the root to a leaf of length d , that path contains a vertex of S distinct from the root.) Hence, if we work on unweighted graphs, in order to keep the set H_d valid and relatively small, we can maintain shortest path trees up to depth d from all vertices using the Even-Shiloach algorithm [10] in $O(nmd)$ total time, and recompute H_d using King's algorithm every $\tilde{O}\left(\frac{n}{d}\right)$ insertions. The total time needed for maintaining the hubs is therefore $\tilde{O}(nmd)$.

Furthermore, we also show how to maintain reliable hubs in a decremental setting. Suppose our goal is to compute a set of hubs that is guaranteed to be valid, which clearly is not the case for the sampled hubs of [27]. We show that if shortest path trees up to depth d are maintained using *dynamic tree* data structures [24, 25], one can recompute a certainly-valid set H_d in $\tilde{O}\left(\frac{n^2}{d}\right)$ time using a Las Vegas algorithm. To this end observe

that one can deterministically *verify* if a set $B \subseteq V$ is a blocker-set of n shortest path trees up to depth d in $\tilde{O}(n \cdot |B|)$ time. Therefore, a hub set H_d can be found by combining the approaches of [27] and [19]: we may sample candidate hub sets of size $\tilde{O}(\frac{n}{d})$ until a blocker-set of the trees is found. The number of trials is clearly constant with high probability.

We further extend this idea and show that the information whether B is a blocker-set of a collection of n shortest path trees up to depth d can be maintained subject to the changes to these trees with only polylogarithmic overhead. Consequently, we can detect when the sampled hub set H_d (for any d) ceases to be a valid hub set in $\tilde{O}(nmd)$ total time. The algorithm may make one-sided error (i.e., say that H_d is no longer a valid hub set when it is actually still good), but the probability of an error is low if we assume that the update sequence does not depend on our random bits. Subsequently we show how to extend this idea to improve the total update time to $\tilde{O}(nm)$. Assume we are given a valid d -hub set H_d . We prove that in order to verify whether H_{6d} is a valid $6d$ -hub set, it suffices to check whether it hits sufficiently long paths between the elements of H_d . We use this observation to maintain a family of reliable hub sets $H_1, H_6, \dots, H_{6^i}, \dots, H_{6^k}$ (where $6^k \leq n$) under edge deletions (or under edge insertions) in $\tilde{O}(nm)$ total time. Using that, we immediately improve the state-of-the-art decremental APSP algorithms of Baswana et al. [4] (for the exact unweighted case) and Bernstein [5] (for the $(1 + \epsilon)$ -approximate case) from Monte Carlo to Las Vegas (but still assuming an oblivious adversary) by only adding a polylogarithmic factor to the total update time bound.

Generalization to weighted digraphs. Adapting the reliable hub sets maintenance (for both described approaches: the incremental one and sample/verify) to *weighted* digraphs turns out to be far from trivial. This is much different from the sampling approach of Ullman and Yannakakis [27], which works regardless of whether the input graph is weighted or not. The primary difficulty is maintaining all shortest paths consisting of up to d edges. While in the unweighted case the length of a path is equal to the number of edges on this path, this is no longer true in the weighted case.

To bypass this problem we first relax our definition of hubs. For each $u, v \in V$ we require that some $(1 + \epsilon)$ -approximate shortest $u \rightarrow v$ path contains a hub on each subpath consisting of at least $d + 1$ edges. Next, we show that running King's blocker-set algorithm on a set of $(1 + \epsilon)$ -approximate shortest path trees up to depth⁴ d from all vertices of the graph yields a hub set that hits paths approximating the true shortest paths within a factor of $(1 + \epsilon)^{\Theta(\log n)}$. Note that a collection of such trees can be maintained in $\tilde{O}(nmd \log W/\epsilon)$ total time subject to edge insertions, using Bernstein's h -SSSP algorithm [5] with $h = d$.

The $\Theta(\log n)$ exponent in the approximation ratio comes from the following difference between the weighted and unweighted case. In a $(1 + \epsilon)$ -approximate shortest path tree up to depth d , the length of a $u \rightarrow v$ path is no more than $(1 + \epsilon)$ -times the length of the shortest $u \rightarrow v$ path in G that uses at most d edges. However, the $u \rightarrow v$ path in the tree might consist of any number of edges, in particular very few. Pessimistically, all these trees have depth $o(d)$ and their blocker-set is empty, as there is no path of hop-length $\Omega(d)$ that we need to hit. Note that this is an inherent problem, as the fact that we can find a small blocker-set in the unweighted case relies on the property that we want it to hit paths of $\Omega(d)$ edges.

⁴ In such a tree (see Definition 23), which is a subgraph of G , for all $v \in V$, the path from the source s to v has length not exceeding $(1 + \epsilon)$ times the length of a shortest out of $s \rightarrow v$ paths in G that use no more than d edges; however the tree path itself can have arbitrary number of edges.

Luckily, a deeper analysis shows that our algorithm can still approximate the length of a $s \rightarrow v$ path. Roughly speaking, we split the $s \rightarrow v$ path P into two subpaths of $d/2$ edges. If each of these two subpaths are approximated in the h -SSSP data structures by paths of less than $d/4$ edges, we replace the P by the concatenation of the two approximate paths from the h -SSSP data structures. This way, we get a path that can be longer by a factor of $(1 + \epsilon)$, but whose hop-length is twice smaller. By repeating this process $O(\log n)$ times we obtain a path of constant hop-length whose length is at most $(1 + \epsilon)^{\Theta(\log n)}$ larger than the length of P . The overall approximation ratio is reduced to $(1 + \epsilon)$ by scaling ϵ by a factor of $\Theta(\log n)$.

Deterministic incremental all-pairs shortest paths. We now show how to apply our results on reliable hubs to obtain an improved algorithm for incremental all-pairs shortest paths problem in weighted digraphs. We give a deterministic incremental algorithm maintaining all-pairs $(1 + \epsilon)$ -approximate distance estimates in $\tilde{O}(mn^{4/3} \log W/\epsilon)$ total time.

Let us now give a brief overview of our algorithm in the unweighted case. First, we maintain the set of hubs H_d under edge insertions as described above in $\tilde{O}(nmd)$ total time. Second, since the set H_d changes and each vertex of the graph may eventually end up in H_d , we cannot afford maintaining shortest path trees from all the hubs (which is done in most algorithms that use hubs). Instead, we use the folklore $\tilde{O}(n^3/\epsilon)$ total time incremental $(1 + \epsilon)$ -approximate APSP algorithm [5, 19] to compute distances between the hubs. Specifically, we run it on a graph whose vertex set is H_d and whose edges represent shortest paths between hubs of hop-lengths at most d . These shortest paths are taken from the shortest path trees up to depth d from all $v \in V$ that are required for the hub set maintenance. We reinitialize the algorithm each time the set H_d is recomputed. This allows us to maintain approximate pairwise distances between the hubs at all times in $\tilde{O}\left(m(n/d)^2/\epsilon\right)$ total time.

Finally, we show how to run a dynamic algorithm on top of a *changing* set of hubs by adapting the shortcut edges technique of Bernstein [5]. Roughly speaking, the final estimates are maintained using $(1 + \epsilon)$ -approximate shortest path trees [5] up to depth $O(d)$ from all vertices v on graph G augmented with shortcuts from v to H_d and from H_d to v . This poses some technical difficulties as the set of shortcuts is undergoing both insertions (when a hub is added) and deletions (when the entire set of hubs is recomputed from scratch). However, one can note that in the incremental setting the shortcuts that no longer approximate the distances between their endpoints do not break the approximation guarantee of our algorithm. Eventually, we use shortcuts between all pairs of vertices of G but only some of them are guaranteed (and sufficient) to be up to date at any time. The total time cost of maintaining this component is $\tilde{O}(nmd/\epsilon)$. Setting $d = \tilde{O}(n^{1/3})$ gives the best update time.

It is natural to wonder if this approach could be made to work in the decremental setting. There are two major obstacles. First, it is unclear whether one can deterministically maintain a valid set of hubs under deletions so that only $O(1)$ vertices (in amortized sense) are added to the hub set after each edge deletion. Note that in extreme cases, after a single edge deletion the set of hubs may have to be extended with polynomially many new vertices. Second, all algorithms using the above approach of introducing shortcuts from and to hubs also maintain a decremental shortest path data structure on a graph consisting of the edges of the original graph and shortcut edges representing distances between the hubs. If hubs were to be added, the graph maintained by the data structure would undergo both insertions (of shortcuts) and deletions (of edges of the original graph) which would make this a much harder, fully dynamic problem. Some earlier works dealt with a similar issue by ignoring some “inconvenient” edge insertions [14] or showing that the insertions are well-behaved [6]. However, these approaches crucially depended on the graph being undirected.

1.2 Related Work

The dynamic graph problems on digraphs are considerably harder than their counterparts on undirected graphs. An extreme example is the dynamic reachability problem, that is, transitive closure on directed graphs, and connectivity on undirected graphs. While there exist algorithms for undirected graphs with polylogarithmic query and update times [17, 28, 26, 16, 18], in the case of directed graphs the best known algorithm with polylogarithmic query time has an update time of $O(n^2)$ [23, 7, 20]. In addition, a combinatorial algorithm with an update time of $O(n^{2-\epsilon})$ is ruled out under Boolean matrix multiplication conjecture [1].

In 2003, in a breakthrough result Demetrescu and Italiano gave a fully dynamic, exact and deterministic algorithm for APSP in weighted directed graphs [8]. The algorithm handles updates in $\tilde{O}(n^2)$ amortized time and maintains the distance matrix explicitly. The bound of $O(n^2)$ is a natural barrier as a single edge insertion or deletion may change up to $\Omega(n^2)$ entries in the distance matrix. For dynamic APSP in digraphs there exists faster algorithms with polylogarithmic query time, all of which work in incremental or decremental setting:

- Ausiello et al. [3] gave a deterministic incremental algorithm for exact distances in unweighted digraphs that handles updates in $\tilde{O}(n^3)$ total time.
- Baswana et al. [4] solved the same problem in the decremental setting with a Monte Carlo algorithm with $\tilde{O}(n^3)$ total update time.
- Bernstein [5] gave a Monte Carlo algorithm for $(1 + \epsilon)$ -approximate distances in weighted graphs (with weights in $[1, W]$) with $\tilde{O}(nm \log W/\epsilon)$ total update time. The algorithm works both in the incremental and decremental setting.
- Finally, deterministic partially-dynamic (both incremental and decremental) algorithms for APSP in directed graphs with $\tilde{O}(n^3 \log W/\epsilon)$ total update time can be obtained by combining the results of [19] and [5].

The algorithms of Baswana et al. [4] and Bernstein [5] both use sampled hubs and thus require the *oblivious adversary* assumption. We highlight that in the class of deterministic algorithms, the best known results have total update time $\tilde{O}(n^3)$ [3, 5], even if we only consider sparse unweighted graphs in incremental or decremental setting and allow $(1 + \epsilon)$ approximation. In the incremental setting, for not very dense graphs, when $m = O(n^{5/3-\epsilon})$, our algorithm improves this bound to $\tilde{O}(mn^{4/3})$.

Organization of the paper. In Section 2 we fix notation, review some of the existing tools that we use and give a formal definition of hubs. Section 3 describes the hub set maintenance for incremental unweighted digraphs and our $(1 + \epsilon)$ -approximate incremental algorithm for sparse graphs. In Section 4 we show a faster Las Vegas algorithm for computing reliable hubs and further extend it to maintain reliable hub sets in the partially dynamic setting. There we also sketch how to use it in order to improve the state-of-the-art decremental APSP algorithms from Monte Carlo to Las Vegas randomized. Finally, in Section 5 we sketch how to adapt the hub set maintenance algorithms of Sections 3 and 4, so that they work on weighted graphs. Due to limited space, many proofs and details can only be found in the full version of this paper.

2 Preliminaries

In this paper we deal with *directed* graphs. We write $uv \in E(G)$ when referring to edges of G and use $w_G(uv)$ to denote the weight of uv . If G is unweighted, then $w_G(e) = 1$ for each $e \in E$. For weighted graphs, $w_G(e)$ can be any real number from the interval $[1, W]$. For simplicity, in this paper we assume that W is an input parameter given beforehand. If $uv \notin E$, we assume

$w_G(uv) = \infty$. We define the union $G \cup H$ to be the graph $(V(G) \cup V(H), E(G) \cup E(H))$ with weights $w_{G \cup H}(uv) = \min(w_G(uv), w_H(uv))$ for each $uv \in E(G \cup H)$. For an edge $e = uv$, we write $G + e$ to denote $(V(G) \cup \{u, v\}, E(G) \cup \{e\})$. The *reverse graph* G^R is defined as $(V(G), \{xy : yx \in E(G)\})$ and $w_{G^R}(xy) = w_G(yx)$.

A sequence of edges $P = e_1 \dots e_k$, where $k \geq 1$ and $e_i = u_i v_i \in E(G)$, is called a $u \rightarrow v$ path in G if $u = u_1$, $v_k = v$ and $v_{i-1} = u_i$ for each $i = 2, \dots, k$. We sometimes view a path P in G as a subgraph of G with vertices $\{u_1, \dots, u_k, v\}$ and edges $\{e_1, \dots, e_k\}$ and write $P \subseteq G$. The *hop-length* $|P|$ is defined as $|P| = k$. The *length* of the path $\ell(P)$ is defined as $\ell(P) = \sum_{i=1}^k w_G(e_i)$. If G is unweighted, then we clearly have $|P| = \ell(P)$. For convenience, we sometimes consider a single edge uv a path of hop-length 1. It is also useful to define a length-0 $u \rightarrow u$ path to be the graph $(\{u\}, \emptyset)$. If P_1 is a $u \rightarrow v$ path and P_2 is a $v \rightarrow w$ path, we denote by $P_1 \cdot P_2$ (or simply $P_1 P_2$) a path $P_1 \cup P_2$ obtained by concatenating P_1 with P_2 .

A digraph T is called an *out-tree over V rooted in r* if $v \in V(T) \subseteq V$, $|E(T)| = |V(T)| - 1$ and for all $v \in V(T)$ there is a unique path $T[v]$ from r to v . The depth $\text{dep}_T(v)$ of a vertex $v \in V(T)$ is defined as $|T[v]|$. The depth of T is defined as $\max_{v \in V(T)} \{\text{dep}_T(v)\}$. Each non-root vertex of an out-tree has exactly one incoming edge. For $v \in V(T) \setminus \{r\}$ we call the other endpoint of the incoming edge of v the *parent* v and write $\text{par}_T(v)$ when referring to it.

The *distance* $\delta_G(u, v)$ between the vertices $u, v \in V(G)$ is the length of the shortest $u \rightarrow v$ path in G , or ∞ , if no $u \rightarrow v$ path exists in G . We define $\delta_G^k(u, v)$ to be the length of the shortest path from u to v among paths of at most k edges. Formally, $\delta_G^k(u, v) = \min\{\ell(P) : u \rightarrow v = P \subseteq G \text{ and } |P| \leq k\}$. We sometimes omit the subscript G and write $w(uv)$, $\delta(u, v)$, $\delta^k(u, v)$ etc. instead of $w_G(u, v)$, $\delta_G(u, v)$, $\delta_G^k(u, v)$, etc., respectively.

We say that a graph G is *incremental*, if it only undergoes edge insertions and edge weight decreases. Similarly, we say that G is *decremental* if it undergoes only edge deletions and edge weight increases. We say that G is *partially dynamic* if it is either incremental or decremental. For a dynamic graph G we denote by n the maximum value of $|V|$ and by m the maximum value of $|E|$ throughout the whole sequence of updates.

We denote by Δ the total number of updates a dynamic graph G is subject to. If G is unweighted, then clearly $\Delta \leq m$ and in fact we assume $\Delta = m$, which allows us to simplify the analyses. For weighted digraphs, on the other hand, since the total number of weight increases/decreases that an edge is subject to is unlimited, Δ may be much larger than m . As a result, it has to be taken into account when analyzing the efficiency of our algorithms.

We call a partially-dynamic $(1 + \epsilon)$ -approximate APSP problem on weighted graphs *restricted* if the edge weights of G are of the form $(1 + \epsilon)^i$ for $i \in [0, \lceil \log_{1+\epsilon} W \rceil] \cap \mathbb{N}$ at all times and additionally each update is required to actually change the edge set or change the weight of some existing edge. Consequently, observe that in the restricted problem we have $\Delta \leq m \cdot (\lceil \log_{1+\epsilon} W \rceil + 2)$. In the following we concentrate on the restricted problem. This is without much loss of generality – we provide a reduction in the full version of the paper.

Partially-dynamic single-source shortest path trees.

► **Definition 1.** Let $G = (V, E)$ be an unweighted digraph and let $s \in V$. Let $d > 0$ be an integer. We call an out-tree $T \subseteq G$ rooted in s a *shortest path tree from s up to depth d* if: (a) for any $v \in V$, $v \in V(T)$ iff $\delta_G(s, v) \leq d$, and (b) for any $v \in V(T)$, $\delta_T(s, v) = \delta_G(s, v)$.

► **Theorem 2 (Even-Shiloach tree [10, 15]).** Let $G = (V, E)$ be an unweighted graph subject to partially dynamic edge updates. Let $s \in V$ and let $d \geq 1$ be an integer. Then, a shortest path tree from s up to depth d can be explicitly maintained⁵ in $O(md)$ total time.

⁵ By this we mean that the algorithm outputs all changes to the edge set of the maintained tree.

► **Theorem 3** (*h*-SSSP [5]). Let $G = (V, E)$ be a weighted digraph. Let $s \in V$ and let $h \geq 1$ be an integer. There exists a partially dynamic algorithm explicitly maintaining $(1 + \epsilon)$ -approximate distance estimates $\delta'(s, v)$ satisfying $\delta_G(s, v) \leq \delta'(s, v) \leq (1 + \epsilon)\delta_G^h(s, v)$ for all $v \in V$. The total update time of the algorithm $O(mh \log n \log(nW)/\epsilon + \Delta)$.

Hubs and how to compute them.

► **Definition 4.** Let $G = (V, E)$ be a directed graph. Let $B \subseteq V$ and let $d > 0$ be an integer. We say that a path P in G is (B, d) -covered if it can be expressed as $P = P_1 \dots P_k$, where $P_i = u_i \rightarrow v_i$, $|P_i| \leq d$ for each $i = 1, \dots, k$, and $u_i \in B$ for each $i = 2, \dots, k$.

We now define a blocker set, slightly modifying a definition by King [19].

► **Definition 5.** Let V be a vertex set and let d be a positive integer. Let $B \subseteq V$ and let T be a rooted tree over V of depth no more than d . We call B a (T, d) -blocker set if for each $v \in V(T)$ such that $\text{dep}_T(v) = d$, either v or one of its ancestors in T belongs to B .

Let \mathcal{T} be a collection of rooted trees over V of depth no more than d . We call B a (\mathcal{T}, d) -blocker set if B is a (T, d) -blocker set for each $T \in \mathcal{T}$.

► **Lemma 6.** Let V be a vertex set of size n . Let d be a positive integer. Let \mathcal{T} be a collection of rooted trees over V of depth at most d . Then, a (\mathcal{T}, d) -blocker set of size $O(\frac{n}{d} \log n)$ can be computed deterministically in $O(n \cdot (|\mathcal{T}| + n) \log n)$ time.

► **Definition 7.** Let $G = (V, E)$ be a directed graph and let $d > 0$ be an integer. A set $H_d \subseteq V$ is called a d -hub set of G if for every $u, v \in V$ such that $\delta_G(u, v) < \infty$, there exists some shortest $u \rightarrow v$ path that is (H_d, d) -covered.

► **Lemma 8.** Let $G = (V, E)$ be a directed unweighted graph and let $d > 0$ be an integer. Suppose we are given a collection $\mathcal{T} = \{T_v : v \in V\}$ of shortest path trees up to depth d from all vertices of G . Let B be a (\mathcal{T}, d) -blocker set. Then B is a $2d$ -hub set of G .

Deterministic incremental algorithm for dense graphs.

► **Theorem 9** ([19]+[5]). There exist an incremental algorithm maintaining all-pairs $(1 + \epsilon)$ -approximate distance estimates of a digraph in $\tilde{O}(n^3 \log(W)/\epsilon) + O(\Delta)$ total time.

As mentioned before, the above theorem basically follows by combining the partially dynamic transitive closure algorithm of King [19] with Bernstein's *h*-SSSP algorithm (Theorem 3) for $h = 2$.

3 Deterministic Incremental Algorithm for Sparse Graphs

In this section we present our deterministic incremental algorithm with $\tilde{O}(mn^{4/3}/\epsilon)$ total update time. We first observe that whenever an edge xy is added, the set of hubs may be “fixed” by extending it with both x and y .

► **Lemma 10.** Let $G = (V, E)$ be a directed unweighted graph. Let H_d be a d -hub set of G . Let $x, y \in V$ be such that $xy \notin E$. Then $H'_d = H_d \cup \{x, y\}$ is a d -hub set of $G' = G + xy$.

Let $d > 1$ be an even integer and let $\epsilon_1, 0 < \epsilon_1 < \epsilon$ be a real number, both to be set later. Our data structure consists of several components. Each subsequent component builds upon the previously defined components only.

Exact shortest paths between nearby vertices. The data structure maintains two collections $\mathcal{T}^{\text{from}} = \{T_v^{\text{from}} : v \in V\}$ and $\mathcal{T}^{\text{to}} = \{T_v^{\text{to}} : v \in V\}$ of shortest path trees up to depth $\frac{d}{2}$ in G and G^{R} , resp. By Theorem 2, each tree of $\mathcal{T}^{\text{from}} \cup \mathcal{T}^{\text{to}}$ can be maintained under edge insertions in $O(md)$ total time. The total time spent in this component is hence $O(nmd)$.

The hubs. A d -hub set H_d of both G and G^{R} such that $|H_d| = O(\frac{n}{d} \log n)$ is maintained at all times, as follows. Initially, H_d is computed in $O(n^2 \log n)$ time using Lemma 6 and the trees of $\mathcal{T}^{\text{from}} \cup \mathcal{T}^{\text{to}}$ (see Lemma 8). Next, the data structure operates in phases. Each phase spans $f = \Theta(\frac{n}{d} \log n)$ consecutive edge insertions. When an edge xy is inserted, its endpoints are inserted into H_d . By Lemma 10, this guarantees that H_d remains a d -hub set of both G and G^{R} after the edge insertion. Once f edges are inserted in the current phase, the phase ends and the hub set H_d is recomputed from scratch, again using Lemma 6. Observe that the size of $|H_d|$ may at most triple within each phase.

The total time spent on maintaining the set H_d is clearly $O(\frac{m}{f} \cdot n^2 \log n) = O(nmd)$.

Approximate shortest paths between the hubs. In each phase, we maintain a *weighted* graph $A = (H_d, E_A)$, where $E_A = \{uv : u, v \in H_d, \delta_{G^{\text{R}}}(u, v) \leq d\}$ and $w_A(uv) = \delta_{T_u^{\text{to}}}(u, v) = \delta_{G^{\text{R}}}(u, v) \leq d$. Observe that during each phase, the graph A is in fact incremental. We can thus maintain $(1 + \epsilon_1)$ -approximate distance estimates $\delta'_A(u, v)$ for all $u, v \in H_d$ in $\tilde{O}(|H_d|^3/\epsilon_1) = \tilde{O}((\frac{n}{d})^3/\epsilon_1)$ total time per phase, using a data structure \mathcal{D}_A of Theorem 9.⁶

Summing over all phases, the total time spent on maintaining the $(1 + \epsilon')$ -approximate distances estimates of the graph A is $\tilde{O}(m(\frac{n}{d})^2/\epsilon_1)$.

► **Lemma 11.** For any $u, v \in H_d$, $\delta_{G^{\text{R}}}(u, v) = \delta_A(u, v)$.

By the above lemma, for each $u, v \in H_d$ we actually have $\delta'_A(u, v) \leq (1 + \epsilon_1)\delta_{G^{\text{R}}}(u, v)$.

Shortcuts to hubs. For each $u \in V$, let S_u be a graph on V with exactly n edges $\{uv : v \in V\}$ satisfying $w_{S_u}(u, v) \geq \delta_{G^{\text{R}}}(u, v)$ for all $v \in V$, and additionally $w_{S_u}(u, v) \leq (1 + \epsilon_1)\delta_{G^{\text{R}}}(u, v)$ if u, v both currently belong to H_d . The edges between vertices of H_d are the only ones that our algorithm needs to compute approximate distances. For other edges we only need to make sure they will not cause the algorithm to underestimate the distances.

Observe that the graphs S_u can be maintained using the previously defined components as follows. First, they are initialized so that their edges are all infinite-weight. Whenever the data structure \mathcal{D}_A changes (or initializes) some of its estimates $\delta'_A(u, v) \leq (1 + \epsilon_1)\delta_{G^{\text{R}}}(u, v)$, we perform $w_{S_u}(u, v) := \min(w_{S_u}(u, v), \delta'_A(u, v))$. This guarantees that the invariants posed on S_u are always satisfied and S_u is incremental. The total number of updates to all graphs S_u is equal to the number of estimate updates made by \mathcal{D}_A and thus can be neglected.

For $u \in V$, we set up a h -SSSP data structure \mathcal{D}_u of Theorem 3 for the graph $G^{\text{R}} \cup S_u$ with source vertex u and $h = d + 1$. Hence, \mathcal{D}_u maintains distance estimates $\delta'(u, v)$ such that $\delta'(u, v) \leq (1 + \epsilon')\delta_{G^{\text{R}} \cup S_u}^{d+1}(u, v)$. As the graph $G^{\text{R}} \cup S_u$ is incremental and has $O(m)$ edges, the total time that \mathcal{D}_u needs to operate is $\tilde{O}(md/\epsilon_1) + O(\Delta_u)$, where Δ_u is the total number of updates to $G^{\text{R}} \cup S_u$. Summing the update times for all data structures \mathcal{D}_u , we obtain

⁶ Technically speaking, the total update time of the data structure of Theorem 9 is $\tilde{O}(n^3/\epsilon') + O(\Delta)$. However, all updates to \mathcal{D}_A arise when some previous component updates its explicitly maintained estimates, so the Δ term is asymptotically no more than the total update time of the previously defined components and can be charged to those. In the following, we omit Δ terms like this without warnings.

$\tilde{O}(nmd/\epsilon_1) + O(\sum_{v \in V} \Delta_u)$ total time. Note that $\sum_{u \in V} \Delta_u$ equals nm plus the number of updates to the graphs S_u , which can be charged to the operating cost of data structure \mathcal{D}_A , as argued before. We conclude that the total update time of all \mathcal{D}_u is $\tilde{O}(nmd/\epsilon_1)$.

Observe that a shortest $u \rightarrow v$ path in G^R , where $u \in H_d$ and $v \in V$ is approximated by a path in $G^R \cup S_u$ consisting of at most $d+1$ edges. The first edge belongs to S_u and “jumps” to some hub. The latter (at most d) edges belong to G^R . This is formalized as follows.

► **Lemma 12.** *Let $u \in H_d$ and $v \in V$. Then $\delta_{G^R \cup S_u}^{d+1}(u, v) \leq (1 + \epsilon_1)\delta_{G^R}(u, v)$.*

By the above lemma, we conclude that for $u \in H_d, v \in V$, the estimate $\delta'(u, v)$ produced by the data structure \mathcal{D}_v satisfies $\delta'(u, v) \leq (1 + \epsilon_1)\delta_{G^R \cup S_u}^{d+1}(u, v) \leq (1 + \epsilon_1)^2\delta_{G^R}(u, v)$.

All-pairs approximate shortest paths. We maintain another set of shortcut graphs R_u , for $u \in V$. Again R_u has exactly n edges $\{uv : v \in V\}$ whose weights satisfy $w_{R_u}(uv) \geq \delta_G(u, v)$ for all v and $w_{R_u}(uv) \leq (1 + \epsilon_1)^2\delta_G(u, v)$ if $v \in H_d$. Each graph R_u is maintained using the previously defined data structures \mathcal{D}_v . Initially all weights of R_u are infinite. Whenever some \mathcal{D}_v changes the estimate $\delta'(v, u)$, we set $w_{R_u}(uv) := \min(w_{R_u}(uv), \delta'(v, u))$. Since for $v \in H_d$ we have $\delta'(v, u) \leq (1 + \epsilon_1)^2\delta_{G^R}(v, u)$, equivalently, $\delta'(v, u) \leq (1 + \epsilon_1)^2\delta_G(u, v)$ and we obtain $w_{R_u}(uv) \leq (1 + \epsilon_1)^2\delta_G(u, v)$. Therefore, the graphs R_u are all incremental and the total number of changes they are subject to is no more than the total number of estimate changes made by the data structures $\mathcal{D}_v, v \in V$. Thus, we may neglect the cost of actually performing these changes.

Finally, for each $u \in V$ we set up a h -SSSP data structure \mathcal{D}'_u of Theorem 3 on graph $G \cup R_u$ with source u and $h = d+1$, maintaining $(1 + \epsilon_1)$ -approximate estimates of the values $\delta_{G \cup R_u}^{d+1}(u, \cdot)$. Similarly as was the case for the data structures \mathcal{D}_u of the previous component, as the graphs $G \cup R_u$ are incremental, the total operating time of the h -SSSP instances running on the graphs $G \cup R_u$ is $\tilde{O}(nmd/\epsilon_1)$.

► **Lemma 13.** *Let $u, v \in V$. Then $\delta_{G \cup R_u}^{d+1}(u, v) \leq (1 + \epsilon_1)^2\delta_G(u, v)$.*

By the above lemma, the the distance estimates $\delta''(u, v)$ maintained by the data structure \mathcal{D}'_u , approximate the corresponding distances $\delta_G(u, v)$ within a factor of $(1 + \epsilon_1)^3$.

► **Theorem 14.** *Let G be a directed unweighted graph. There exists a deterministic incremental algorithm maintaining $(1 + \epsilon)$ -approximate distance estimates between all pairs of vertices of G in $\tilde{O}(mn^{4/3}/\epsilon)$ total time.*

4 Partially-Dynamic Verification of a Sampled Hub Set

In this section we show how to maintain the information whether a sampled set remains a hub set of an unweighted digraph G subject to partially dynamic updates. For simplicity, assume that G is decremental (the incremental case, being somewhat easier, can be handled similarly). We start by showing how a reliable hub set can be found if we are given shortest path trees up to depth d from all vertices of G , stored in dynamic tree data structures.

► **Lemma 15** ([27, 29]). *Let V be a vertex set of size n and let $d > 0$ be an integer. Let \mathcal{T} be a collection of rooted trees of depth no more than d , whose vertex sets are subsets of V .*

Let $c > 1$ be some constant. Let B be a random subset of V of size $\min(\lceil c \frac{n}{d} \ln n \rceil, n)$. Then, B is a (\mathcal{T}, d) -blocker set with probability at least $\max(0, 1 - |\mathcal{T}|/n^{c-1})$.

The following data structure is an easy application of the data structure of Tarjan [25].

► **Lemma 16.** *Let V be some set of n vertices. Let F be a forest of (initially single-vertex) rooted out-trees over V such that the vertex sets of the individual trees of F form a partition of V . For $v \in V$, let $T_v \in F$ denote the unique tree of F containing v .*

There exists a data structure for dynamically maintaining F (initially consisting of n 1-vertex trees) and supporting the following operations in $O(\log n)$ time each:

1. **parent**(v): *if v is not the root of T_v , return its parent. Otherwise return **nil**.*
2. **link**(u, v): *assuming $T_u \neq T_v$ and that u is the root of T_u , make u a child of v in T_v .*
3. **cut**(v): *assuming v is not the root of T_v , split T_v into two trees by removing the edge between v and its parent.*
4. **depth**(v): *return the depth of the tree T_v .*

► **Lemma 17.** *Let V be a vertex set, $n = |V|$, and let $d > 0$ be integral. Let \mathcal{T} be a collection of rooted trees over V of depth no more than d , where $|\mathcal{T}| = O(\text{poly } n)$. Suppose each $T \in \mathcal{T}$ is given as a separate data structure of Lemma 16 and for each $T \in \mathcal{T}$, $\text{root}(T)$ is known.*

Then, there exists a Las Vegas randomized algorithm computing a (\mathcal{T}, d) -blocker set B of size $O(\frac{n}{d} \log n)$ in $O(|\mathcal{T}| \cdot \frac{n}{d} \cdot \log^2 n)$ time with high probability.

Proof. Let $|\mathcal{T}| = O(n^\alpha)$ for some $\alpha > 0$. The algorithm is to simply repeatedly pick random subsets B of V of size $\min(\lceil (\alpha + 2) \frac{n}{d} \ln n \rceil, n)$ until B succeeds in being a (\mathcal{T}, d) -blocker set. By Lemma 15, for a random B , the probability that this is not the case is at most $\frac{1}{n}$. Hence, the probability that we fail finding a (\mathcal{T}, d) -blocker set after $k = O(1)$ trials is at most $1/n^k$.

We thus only need to show how to verify whether a set B is actually a (\mathcal{T}, d) -blocker set in $O(|\mathcal{T}| \cdot |B| \log n)$ time. Recall that for a single $T \in \mathcal{T}$, if the depth of T is no more than d , then B is a (T, d) -blocker set if the tree T' obtained from T by removing all subtrees rooted in vertices of B , has depth less than d . Consequently, to verify whether B is a (T, d) -blocker set, we take advantage of the fact that T is stored in a data structure of Lemma 16.

We first check whether $r = \text{root}(T) \in B$. If this is the case, B is a (T, d) -blocker set in a trivial way. Otherwise, for each $b \in B$, we store $p_b = \text{parent}(b)$ and perform **cut**(b). Afterwards, one can see that B is a (T, d) -blocker set if and only if $\text{depth}(r) < d$. Finally, we revert all the performed **cut** operations by running **link**(b, p_b) for all $b \in B$.

Clearly, the time needed to verify whether B is a (T, d) -blocker set for any $T \in \mathcal{T}$, is $O(|B| \log n)$. Hence, one can check whether B is a (\mathcal{T}, d) -blocker set in $O(|\mathcal{T}| \cdot |B| \log n)$ time. ◀

Now we move on to the problem of detecting when a sampled set ceases to be a valid hub set of G . In fact, our algorithm will solve a bit more general problem (which is anyway needed for applications, as we will see later), as follows.

Let $|V| = n = a_0 > a_1 > \dots > a_q = 1$ be some sequence of integers such that $a_i \mid a_{i-1}$. For each $i = 0, \dots, q$, let A_i be a random a_i -subset (a subset of size a_i) of V . By Lemmas 8 and 15, each A_i is in fact an $\Theta((n/a_i) \ln n)$ -hub set of G with high probability.

We would like to detect when some A_i ceases to be an $\Theta((n/a_i) \ln n)$ -hub set of G while G undergoes edge deletions. Using this terminology, both state-of-the-art Monte-Carlo randomized algorithms for decremental exact shortest paths [4] and partially-dynamic $(1 + \epsilon)$ -approximate shortest paths [5] (for unweighted digraphs) use randomness only for constructing hub sets A_0, \dots, A_q (they use $a_i = 2^{q-i}$, but in fact any $a_i = c^{q-i}$, where c is a positive integer, would be sufficient for these algorithms to work), valid simultaneously for all versions of the input graph with high probability (the sets A_i satisfy this, as we will later show).

Without loss of generality, we can assume that given the sets A_0, \dots, A_q , the algorithms of [4, 5] proceed deterministically. Suppose we develop an efficient partially dynamic algorithm \mathcal{A} verifying whether each A_i remains a $\Theta((n/a_i) \ln n)$ -hub set of G (i.e., \mathcal{A} is supposed to

detect that some A_i ceases to be a $\Theta((n/a_i) \ln n)$ -hub set immediately after this happens) and producing *false negatives* with low probability (the algorithm is guaranteed to be correct if it says that all A_i have the desired property but might be wrong saying that some A_i is no longer a hub set). Then, we could use \mathcal{A} to convert the algorithms of [4, 5] into *Las Vegas* algorithms by drawing new sets A_0, \dots, A_q and restarting the respective algorithms whenever \mathcal{A} detects (possibly incorrectly) that any of these sets ceases to be a hub set. As this does not happen w.h.p., with high probability the overall asymptotic running time remains unchanged. The remainder of this section is devoted to describing such an algorithm \mathcal{A} .

► **Lemma 18.** *Let $d > 0$ be an integer. Let F be a forest of out-trees of depth no more than d over V . Denote by T_v the unique tree of F containing $v \in V$. Let $B \subseteq V$ be fixed.*

There exists a data structure with update time $O(\log n)$, maintaining the information whether B is a (F, d) -blocker set, subject to updates to F of the following types:

- *cut the subtree rooted in v out of T_v where $v \in V$ and v is not the root of T_v ,*
- *make the tree T_r a child of $v \in T_v$ where $r \in V$ is the root of T_r and $v \notin T_r$,*

The following lemma says that in order to test whether a given set of vertices is a $6d$ -hub set it suffices to test the hub set property for paths starting in vertices of a d -hub set.

► **Lemma 19.** *Let $G = (V, E)$ be a directed unweighted graph. Let H_d be a d -hub set of G . Suppose we are given two collections $\mathcal{T}^{\text{from}} = \{T_v^{\text{from}} : v \in H_d\}$, $\mathcal{T}^{\text{to}} = \{T_v^{\text{to}} : v \in H_d\}$ of shortest path trees up to depth d from all vertices of H_d in G and G^{R} , respectively.*

Let B be a $(\mathcal{T}^{\text{from}} \cup \mathcal{T}^{\text{to}}, d)$ -blocker set. Then B is a $6d$ -hub set of G .

Observe that by Lemma 15, there exists an integral constant $z > 0$, such that for any fixed collection of trees \mathcal{T} of depth no more than $z \cdot \frac{n}{a_i} \lceil \ln n \rceil$, where $|\mathcal{T}| = O(n^3)$, A_i is a $(\mathcal{T}, z \cdot \frac{n}{a_i} \lceil \ln n \rceil)$ -blocker set with high probability. For $i = 0, \dots, q$, set $d_i = z \cdot \frac{n}{a_{i+1}} \lceil \ln n \rceil$ where $a_{q+1} = 1$. Suppose G undergoes partially dynamic updates. For each $i = 1, \dots, q$, and $v \in V$ let $T_{i,v}^{\text{from}}$ ($T_{i,v}^{\text{to}}$) denote the shortest path tree that the algorithm of Theorem 2 would maintain for $d = d_{i-1}$ and source v in G (in G^{R} , respectively). Note that how the trees $T_{i,v}^{\text{from}}$ and $T_{i,v}^{\text{to}}$ evolve depends only on the sequence of updates to G (which, by the oblivious adversary assumption, does not depend on sets A_0, \dots, A_q in any way) and the details of the deterministic algorithm of Theorem 2. Since only $O(mn) = O(n^3)$ different trees appear in $\{T_{i,v}^{\text{from}} : v \in V\} \cup \{T_{i,v}^{\text{to}} : v \in V\}$ throughout all updates, A_i remains a $(\{T_{i,v}^{\text{from}} : v \in V\} \cup \{T_{i,v}^{\text{to}} : v \in V\}, d_{i-1})$ -blocker set throughout the whole sequence of updates with high probability, by Lemma 15.

Let $\mathcal{T}_i^{\text{from}} = \{T_{i,v}^{\text{from}} : v \in A_{i-1}\}$ and $\mathcal{T}_i^{\text{to}} = \{T_{i,v}^{\text{to}} : v \in A_{i-1}\}$, i.e., $\mathcal{T}_i^{\text{from}}$ ($\mathcal{T}_i^{\text{to}}$) contains only trees with roots from a subset $A_{i-1} \subseteq V$. However A_i being a blocker set of such a collection of trees will turn out sufficient for our needs. Clearly, since we have $\mathcal{T}_i^{\text{from}} \cup \mathcal{T}_i^{\text{to}} \subseteq \{T_{i,v}^{\text{from}} : v \in V\} \cup \{T_{i,v}^{\text{to}} : v \in V\}$, by the above claim, A_i in fact remains a $(\mathcal{T}_i^{\text{from}} \cup \mathcal{T}_i^{\text{to}}, d_{i-1})$ -blocker set throughout the whole sequence of updates with high probability.

Now, let $q = \lceil \log_6 n \rceil$ and for $i = 1, \dots, q$ set $a_i = 6^{q-i}$. To verify whether each A_i remains a d_i -hub set subject to partially dynamic updates to G , we proceed as follows. We deterministically maintain the trees $\bigcup_{i=1}^q (\mathcal{T}_i^{\text{from}} \cup \mathcal{T}_i^{\text{to}})$ subject to partially dynamic updates to G using Theorem 2. The total number of changes these trees are subject to throughout the whole sequence of updates is $O(\sum_{i=1}^q a_{i-1} \cdot m \cdot d_{i-1}) = O(\sum_{i=1}^q a_{i-1} \cdot m \cdot \frac{n}{a_i} \ln n) = O(nm \log n \cdot \sum_{i=1}^q \frac{a_{i-1}}{a_i}) = \tilde{O}(nm)$.

We additionally store each tree $T_{i,v}^{\text{from}}$ (and $T_{i,v}^{\text{to}}$), for $v \in A_{i-1}$, in a data structure of Lemma 18 with $B = A_i$. Whenever the data structure of Theorem 2 updates some tree, the update is repeated in the corresponding data structure of Lemma 18. Consequently, the total time needed to maintain these additional data structures is $\tilde{O}(nm \cdot \sum_{i=1}^q \frac{a_{i-1}}{a_i}) = \tilde{O}(nm)$.

After each update we can detect whether each A_i is still a $(\mathcal{T}_i^{\text{from}} \cup \mathcal{T}_i^{\text{to}}, d_{i-1})$ -blocker set in $O(\sum_i^q |A_{i-1}| \log n) = \tilde{O}(n)$ time by querying the relevant data structures of Lemma 18⁷ storing $\mathcal{T}_i^{\text{from}} \cup \mathcal{T}_i^{\text{to}}$. By Lemma 19, a simple inductive argument shows that if this is the case, each A_i is a d_i -hub set of both G and G^R . Hence, verifying all A_1, \dots, A_q while G evolves takes $\tilde{O}(mn)$ total time. The algorithm terminates when it turns out that some A_i is no longer a $(\mathcal{T}_i^{\text{from}} \cup \mathcal{T}_i^{\text{to}}, d_{i-1})$ -blocker set. However, recall that this happens only with low probability, regardless of whether A_i actually ceases to be a d_i hub set or not. We have proved the following.

► **Theorem 20.** *Let G be an unweighted digraph. Let $q = \lceil \log_6 n \rceil$. For $i = 0, \dots, q$, let A_i be a random 6^{q-i} -subset of V . One can maintain the information whether each A_i is a $\Theta(6^i \ln n)$ -hub set of G , subject edge deletions issued to G , in $\tilde{O}(nm)$ total time.*

The algorithm might produce false negatives with low probability.

By plugging in the hubs of Theorem 20 into the algorithms of [4, 5], we obtain the following.

► **Corollary 21.** *Let G be an unweighted digraph. There exists a Las Vegas randomized decremental algorithm maintaining exact distance between all pairs of vertices of G with $\tilde{O}(n^3)$ total update time w.h.p. It assumes an adversary oblivious to the random bits used.*

► **Corollary 22.** *Let G be an unweighted digraph. There exists a Las Vegas randomized decremental algorithm maintaining $(1 + \epsilon)$ -approximate distance estimates between all pairs of vertices of G in $\tilde{O}(nm/\epsilon)$ total time w.h.p. The algorithm assumes an oblivious adversary.*

5 Approximate Shortest Paths for Weighted Graphs

In this section we give key definitions used to generalize the reliable hub maintenance algorithms to weighted graphs, at the cost of $(1 + \epsilon)$ -approximation. Then, we state the main theorem (Theorem 26) relating blocker sets in $(1 + \epsilon)$ -approximate shortest path trees to the approximate hub sets. Finally, we explain briefly how to incorporate these tools into our improved dynamic APSP algorithms in order to generalize them to weighted graphs.

► **Definition 23.** *Let $G = (V, E)$ be a weighted digraph and let $s \in V$ be a source vertex. Let d be a positive integer. An out-tree $T \subseteq G$ is called a $(1 + \epsilon)$ -approximate shortest path tree from s up to depth d , if T is rooted at s and for any $v \in V$ such that $\delta_G^d(s, v) < \infty$, we have $v \in V(T)$ and $\delta_T(s, v) \leq (1 + \epsilon)\delta_G^d(s, v)$.*

► **Definition 24.** *Let $G = (V, E)$ be directed and let $d > 0$ be an integer. A set $H_d^\epsilon \subseteq V$ is called an $(1 + \epsilon)$ -approximate d -hub set of G if for every $u, v \in V$ such that $\delta_G(u, v) < \infty$, there exists a path $P = u \rightarrow v$ in G such that $\ell(P) \leq (1 + \epsilon)\delta_G(u, v)$ and P is (H_d^ϵ, d) -covered.*

We also extend the definition of a (T, d) -blocker set to trees of depth more than d .

► **Definition 25.** *Let V be a vertex set and let $d > 0$ be an integer. Let T be a rooted tree over V . Define T^d to be the set of all maximal subtrees of T of depth no more than d , rooted in non-leaf vertices $x \in V(T)$ satisfying $d \mid \text{dep}_T(x)$.*

⁷ The Even-Shiloach algorithm (Theorem 2), apart from maintaining distance labels for all $v \in V$, moves around entire subtrees of the maintained tree T . Hence, in order to ensure that some set B remains a blocker-set of T , it is not sufficient to simply check whether $B \cap V(T[v])$ whenever the Even-Shiloach algorithm changes the distance label of v to d (and, consequently, use a data structure much simpler than that given in Lemma 18).

Then, B is a (T, d) -blocker set if and only if it is a (T^d, d) -blocker set (in terms of Definition 5). Let \mathcal{T} be a collection of rooted trees over V . We call B a (\mathcal{T}, d) -blocker set if and only if B is a (T, d) -blocker for each $T \in \mathcal{T}$.

► **Theorem 26.** Let $G = (V, E)$ be a directed graph and let $d < n$ be an even integer. Let $\mathcal{T}^{\text{from}} = \{T_v^{\text{from}} : v \in V\}$ ($\mathcal{T}^{\text{to}} = \{T_v^{\text{to}} : v \in V\}$) be a collection of $(1 + \epsilon)$ -approximate shortest path trees up to depth $3d$ from all vertices in G (in G^{R} , resp.).

Let $B \subseteq V$ be a $(\mathcal{T}^{\text{from}} \cup \mathcal{T}^{\text{to}}, \frac{d}{2})$ -blocker set. Then B is a $(1 + \epsilon)^p$ -approximate $2dp$ -hub set of G , where $p = \lceil \log_2 n \rceil + 1$.

Recall that our reliable hubs maintenance algorithms for unweighted graphs essentially maintained some shortest path trees up to depth d and either computed their blocker sets using King’s algorithm, or dynamically verified whether the sampled hub sets remain blocker sets of the shortest path trees.

We first replace all shortest path trees up to depth d with $(1 + \epsilon)$ -approximate shortest path trees up to depth d . We use the following extension of Bernstein’s h -SSSP algorithm.

► **Lemma 27.** The h -SSSP algorithm of Theorem 3 can be extended so that it maintains a $(1 + \epsilon)$ -approximate shortest path tree up to depth h from s within the same time bound.

By Theorem 26, by finding blocker sets of approximate shortest path trees (as in Definition 25), we can compute/verify $(1 + \epsilon')^{\Theta(\log n)}$ -approximate $\Theta(d \log n)$ -hub sets as before.

Given appropriate hub sets, all that both our deterministic incremental $(1 + \epsilon)$ -approximate APSP algorithm, and Bernstein’s randomized $(1 + \epsilon)$ -approximate partially dynamic APSP algorithm do, is essentially set up and maintain a “circuit” (i.e., a collection of data structures whose outputs constitute the inputs of other structures) of h -SSSP data structures from the hubs with different parameters h and appropriately set ϵ' . In order to make these algorithms work with our reliable approximate hub sets, we basically need to play with the parameters: increase all h ’s by a polylogarithmic factor, and decrease ϵ' by a polylogarithmic factor.


References

- 1 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.
- 2 Ittai Abraham, Shiri Chechik, and Sebastian Krinninger. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 440–452. SIAM, 2017. doi:10.1137/1.9781611974782.28.
- 3 Giorgio Ausiello, Giuseppe F. Italiano, Alberto Marchetti-Spaccamela, and Umberto Nanni. Incremental Algorithms for Minimal Length Paths. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1990, San Francisco, California, USA.*, pages 12–21, 1990. URL: <http://dl.acm.org/citation.cfm?id=320176.320178>.
- 4 Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. *J. Algorithms*, 62(2):74–92, 2007. doi:10.1016/j.jalgor.2004.08.004.
- 5 Aaron Bernstein. Maintaining Shortest Paths Under Deletions in Weighted Directed Graphs. *SIAM J. Comput.*, 45(2):548–574, 2016. doi:10.1137/130938670.
- 6 Aaron Bernstein and Liam Roditty. Improved Dynamic Algorithms for Maintaining Approximate Shortest Paths Under Deletions. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1355–1365, 2011. doi:10.1137/1.9781611973082.104.

- 7 Camil Demetrescu and Giuseppe F. Italiano. Fully Dynamic Transitive Closure: Breaking Through the $O(n^2)$ Barrier. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 381–389, 2000. doi:10.1109/SFCS.2000.892126.
- 8 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.
- 9 Camil Demetrescu and Giuseppe F. Italiano. Fully dynamic all pairs shortest paths with real edge weights. *J. Comput. Syst. Sci.*, 72(5):813–837, 2006. doi:10.1016/j.jcss.2005.05.005.
- 10 Shimon Even and Yossi Shiloach. An On-Line Edge-Deletion Problem. *J. ACM*, 28(1):1–4, 1981. doi:10.1145/322234.322235.
- 11 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A Subquadratic-Time Algorithm for Decremental Single-Source Shortest Paths. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1053–1072, 2014. doi:10.1137/1.9781611973402.79.
- 12 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 674–683, 2014. doi:10.1145/2591796.2591869.
- 13 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Improved Algorithms for Decremental Single-Source Reachability on Directed Graphs. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 725–736, 2015. doi:10.1007/978-3-662-47672-7_59.
- 14 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Dynamic Approximate All-Pairs Shortest Paths: Breaking the $O(mn)$ Barrier and Derandomization. *SIAM J. Comput.*, 45(3):947–1006, 2016. doi:10.1137/140957299.
- 15 Monika Rauch Henzinger and Valerie King. Fully Dynamic Biconnectivity and Transitive Closure. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 664–672, 1995. doi:10.1109/SFCS.1995.492668.
- 16 Monika Rauch Henzinger and Valerie King. Randomized dynamic graph algorithms with polylogarithmic time per operation. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 519–527, 1995. doi:10.1145/225058.225269.
- 17 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.
- 18 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.
- 19 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.
- 20 Liam Roditty and Uri Zwick. Improved Dynamic Reachability Algorithms for Directed Graphs. *SIAM J. Comput.*, 37(5):1455–1471, 2008. doi:10.1137/060650271.
- 21 Liam Roditty and Uri Zwick. On Dynamic Shortest Paths Problems. *Algorithmica*, 61(2):389–401, 2011. doi:10.1007/s00453-010-9401-5.
- 22 Liam Roditty and Uri Zwick. Dynamic Approximate All-Pairs Shortest Paths in Undirected Graphs. *SIAM J. Comput.*, 41(3):670–683, 2012. doi:10.1137/090776573.
- 23 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.

- 24 Daniel Dominic Sleator and Robert Endre Tarjan. A Data Structure for Dynamic Trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.
- 25 Robert Endre Tarjan. Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm. *Math. Program.*, 77:169–177, 1997. doi:10.1007/BF02614369.
- 26 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.
- 27 Jeffrey D. Ullman and Mihalis Yannakakis. High-Probability Parallel Transitive-Closure Algorithms. *SIAM J. Comput.*, 20(1):100–125, 1991. doi:10.1137/0220006.
- 28 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.
- 29 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.

Min-Cost Flow in Unit-Capacity Planar Graphs

Adam Karczmarz 

Institute of Informatics, University of Warsaw, Poland
a.karczmarz@mimuw.edu.pl

Piotr Sankowski

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

Abstract

In this paper we give an $\tilde{O}((nm)^{2/3} \log C)$ time algorithm for computing min-cost flow (or min-cost circulation) in unit capacity planar multigraphs where edge costs are integers bounded by C . For planar multigraphs, this improves upon the best known algorithms for general graphs: the $\tilde{O}(m^{10/7} \log C)$ time algorithm of Cohen et al. [SODA 2017], the $O(m^{3/2} \log(nC))$ time algorithm of Gabow and Tarjan [SIAM J. Comput. 1989] and the $\tilde{O}(\sqrt{nm} \log C)$ time algorithm of Lee and Sidford [FOCS 2014]. In particular, our result constitutes the first known fully combinatorial algorithm that breaks the $\Omega(m^{3/2})$ time barrier for min-cost flow problem in planar graphs.

To obtain our result we first give a very simple successive shortest paths based scaling algorithm for unit-capacity min-cost flow problem that does not explicitly operate on dual variables. This algorithm also runs in $\tilde{O}(m^{3/2} \log C)$ time for general graphs, and, to the best of our knowledge, it has not been described before. We subsequently show how to implement this algorithm faster on planar graphs using well-established tools: r -divisions and efficient algorithms for computing (shortest) paths in so-called dense distance graphs.

2012 ACM Subject Classification Theory of computation → Network flows

Keywords and phrases minimum-cost flow, minimum-cost circulation, planar graphs

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.66

Related Version A full version of the paper is available at <https://arxiv.org/abs/1907.02274>.

Funding *Adam Karczmarz*: Supported by ERC Consolidator Grant 772346 TUGbOAT and the Polish National Science Centre 2018/29/N/ST6/00757 grant.

Piotr Sankowski: Supported by ERC Consolidator Grant 772346 TUGbOAT.

1 Introduction

The min-cost flow is the core combinatorial optimization problem that now has been studied for over 60 years, starting with the work of Ford and Fulkerson [14]. Classical combinatorial algorithms for this problem have been developed in the 80s. Goldberg and Tarjan [18] showed an $\tilde{O}(nm \log C)$ time weakly-polynomial algorithm for the case when edge costs are integral, and where C is the maximum edge cost. Orlin [31] showed the best-known strongly polynomial time algorithm running in $\tilde{O}(m^2)$ time. Faster weakly-polynomial algorithms have been developed in this century using interior-point methods: Daitch and Spielman [8] gave an $\tilde{O}(m^{3/2} \log(U + C))$ algorithm, and later Lee and Sidford [28] obtained an $\tilde{O}(\sqrt{nm} \log(U + C))$ algorithm, where U is the maximum (integral) edge capacity.

Much attention has been devoted to the unit-capacity case of the min-cost flow problem. Gabow and Tarjan [15] gave a $O(m^{3/2} \log(nC))$ time algorithm. Lee and Sidford [28] matched this bound up to polylogarithmic factors for $m = \tilde{O}(n)$, and improved upon it for larger densities, even though their algorithm solves the case of arbitrary integral capacities. Gabow and Tarjan's result remained the best known bound for more than 28 years – the problem



© Adam Karczmarz and Piotr Sankowski;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 66; pp. 66:1–66:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

witnessed an important progress only very recently. In 2017 an algorithm that breaks the $\Omega(m^{3/2})$ time barrier for min-cost flow problem was given by Cohen et al. [7]. This algorithm runs in $\tilde{O}(m^{10/7} \log C)$ time and is also based on interior-point methods.

It is worth noting that currently the algorithms of [7, 28] constitute the most efficient solutions for the entire range of possible densities (up to polylogarithmic factors) and are also the best-known algorithms for important special cases, e.g., planar graphs or minor-free graphs. Both of these solutions are based on interior point methods and do not shed light on the combinatorial structure of the problem.

In this paper we study the unit-capacity min-cost flow in planar multigraphs. We improve upon [7, 28] by giving the first known $\tilde{O}((mn)^{2/3} \log C) = \tilde{O}(m^{4/3} \log C)$ time algorithm for computing min-cost s, t -flow and min-cost circulation in planar multigraphs.¹ Our algorithm is fully combinatorial and uses the scaling approach of Goldberg and Tarjan [18]. At each scale it implements the classical shortest augmenting path approach similar to the one known from the well-known Hopcroft-Karp algorithm for maximum bipartite matching [19].

Related work. Due to immense number of works on flows and min-cost flows we will not review all of them. Instead we concentrate only on the ones that are relevant to the sparse and planar graph case, as that is the regime where our results are of importance. As already noted above the fastest algorithms for min-cost flows in planar multigraphs are implied by the algorithms for general case. This, however, is not the case for maximum flow problem. Here, the fastest algorithms are based on planar graph duality and reduce the problem to shortest path computations. The undirected s, t -flow problem can be solved in $O(n \log \log n)$ time [22], whereas the directed s, t -flow problem can be solved in $O(n \log n)$ time [3, 11]. Even for the case with multiple source and sinks, a nearly-linear time algorithm is known [4].

These results naturally raise as an open question whether similar nearly-linear bounds could be possible for min-cost flow. Until very recently there has been no progress towards answering this open question. Partial progress was made by devising $\tilde{O}(n^{4/3} \log C)$ time [1] and $\tilde{O}(n^{6/5} \log C)$ time [27] algorithms for min-cost perfect matchings in bipartite planar graphs. Lahn and Raghvendra also give an $\tilde{O}(n^{7/5} \log C)$ time minimum cost perfect matching algorithm for minor-free graphs. These algorithms can be seen as specialized versions of the Gabow-Tarjan's algorithm for the assignment problem [15].

Gabow and Tarjan [15] reduced min-cost flow problem to so-called min-cost perfect degree-constrained subgraph problem on a bipartite multigraph, which they solved by extending their algorithm for minimum cost perfect matching. Hence it seems plausible that the recent algorithm of Lahn and Raghvendra [27] can be extended to solve min-cost flow, since their algorithm builds upon the Gabow-Tarjan algorithm. The reduction presented by Gabow and Tarjan *is not* planarity preserving, though. Nevertheless, min-cost perfect matching problem can be reduced to min-cost flow problem in an efficient and planarity preserving way [29]. The opposite reduction can be done in planarity preserving way as recently shown [33]. However, this reduction is not efficient and produces a graph of quadratic size. Hence, we cannot really take advantage of it.

Overview and comparison to [1, 27]. We concentrate on the *min-cost circulation* problem, which is basically the min-cost flow problem with all vertex demands equal to 0. It is well-known [17] that the min-cost s, t -flow problem can be solved by first computing some s, t -flow

¹ It is known that simple planar graphs have $O(n)$ edges. However, multiple parallel edges (with possibly different costs) are useful in the unit-capacity min-cost flow problem, as they allow us to encode larger edge capacities. Therefore, in this paper we work with planar multigraphs.

f of requested value (e.g., the maximum value), and then finding a min-cost circulation on the residual network G_f . This reduction is clearly planarity-preserving. Since an s, t -flow of any given value (in particular, the maximum value) can be found in a planar graph in nearly-linear time (see [11]), this reduction works in nearly-linear time as well.

Our min-cost circulation algorithm resembles the recent works on minimum cost planar perfect matching [1, 27], in the sense that we simulate some already-good scaling algorithm for general graphs, but implement it more efficiently using the known and well-established tools from the area of planar graph algorithms. However, instead of simulating an existing unit-capacity min-cost flow algorithm, e.g., [15, 17], we use a very simple successive-shortest paths based algorithm that, to the best of our knowledge, has not been described before.

Our algorithm builds upon the cost-scaling framework of Goldberg and Tarjan [18], similarly as the recent simple unit-capacity min-cost flow algorithms of Goldberg et al. [17]. In this framework, a notion of ϵ -optimality of a flow is used. A flow f is ϵ -optimal with respect to a price function p if for any edge $uv = e \in E(G_f)$ we have $c(e) - p(u) + p(v) \geq -\epsilon$.

Roughly speaking, the parameter ϵ measures the quality of a circulation: any circulation is trivially C -optimal wrt. p , whereas any $\frac{1}{n}$ -feasible (wrt. p) circulation is guaranteed to be optimal. The general scheme is to start with a C -optimal circulation, run $O(\log(nC))$ scales that improve the quality of a circulation by a factor of 2, and this way obtain the optimal solution.

We show that a single scale can be solved by repeatedly sending flow along a cheapest $s \rightarrow t$ path in a certain graph G_f'' with a single source s and a single sink t , that approximates the residual graph G_f . Moreover, if we send flow simultaneously along a maximal set of cheapest $s \rightarrow t$ paths at once, like in [12, 19], we finish after $O(\sqrt{m})$ augmentations. However, as opposed to [12, 19], our graph G_f'' is weighted and might have negative edges. We overcome this difficulty as in the classical successive shortest path approach for min-cost flow, by using distances from the previous flow augmentation as a feasible price function that can speed-up next shortest path computation. Our algorithm also retains a nice property² of the Even-Tarjan algorithm that the total length (in terms of the number of edges) of all the used augmenting paths is $O(m \log m)$.

The crucial difference between our per-scale procedure and those of [15, 17] is that we do not “adjust” dual variables $p(v)$ at all while the procedure runs: we only use them to compute G_f'' , and recompute them from scratch in nearly-linear time when the procedure finishes. In particular, the recent results of [1, 27] are quite complicated since, in order to simulate the Gabow-Tarjan algorithm [15], they impose and maintain additional invariants about the duals.

The only bottlenecks of our per-scale procedure are (1) shortest paths computation, (2) picking a maximal set of edge-disjoint $s \rightarrow t$ paths in an unweighted graph³.

We implement these on a planar network using standard methods. Let $r \in [1, n]$ be some parameter. We construct a *dense distance graph* H_f'' (e.g., [13, 16]) built upon an r -division (e.g., [26]) of G_f'' . The graph H_f'' is a compressed representation of the distances in G_f'' with $O(n/\sqrt{r})$ vertices and $O(m)$ edges. Moreover, it can be updated in $\tilde{O}(r)$ time per edge used by the flow. Hence, the total time spent on updating H_f'' is $\tilde{O}(mr)$. As we show, running our per-scale procedure on H_f'' is sufficient to simulate it on G_f'' . Computing distances in a dense distance graph requires $\tilde{O}(n/\sqrt{r})$ time [13, 16]. To complete the construction, we show how

² Gabow-Tarjan algorithm for min-cost bipartite matching has a similar property, which was instrumental for obtaining the recent results on minimum-cost planar bipartite matching [1, 27].

³ This is sometimes called *the blocking flow* problem and can be solved for unit capacities in linear time.

to find a maximal set of edge-disjoint paths in $\tilde{O}(n/\sqrt{r})$ amortized time. To this end, we also exploit the properties of reachability in a dense distance graph, used previously in dynamic reachability algorithms for planar digraphs [21, 24]. This way, we obtain $\tilde{O}(\sqrt{mn}/\sqrt{r} + mr)$ running time per scale. This is minimized roughly when $r = n^{2/3}/m^{1/3}$.

Recall that Lahn and Raghvendra [27] obtained a polynomially better (than ours) bound of $\tilde{O}(n^{6/5} \log C)$, but only for planar min-cost perfect matching problem. To achieve that, they use an additional idea due to Asathulla et al. [1]. Namely, they observe that by introducing vertex weights, one can make augmenting paths avoid edges incident to boundary vertices, thus making the total number of pieces “affected” by augmenting paths truly-sublinear in n . It is not clear how to apply this idea to the min-cost flow problem without making additional assumptions about the structure of the instance, like bounded-degree (then, there are only $O(n/\sqrt{r})$ edges incident to boundary vertices of an r -division), or bounded vertex capacities (so that only $O(1)$ units of flow can go through each vertex; this is satisfied in the perfect matching case). This phenomenon seems not very surprising once we recall that such assumptions lead to better bounds even for general graphs: the best known combinatorial algorithms for min-cost perfect matching run in $O(n^{1/2}m \log(nC))$ time, whereas for min-cost flow in $O(m^{3/2} \log(nC))$ time [15, 17].

Organization of the paper. In Section 2 we introduce the notation and describe the scaling framework of [18]. Next, in Section 3, we describe the per-scale procedure of unit-capacity min-cost flow for general graphs. Finally, in Section 4 we give our algorithm for planar graphs. Due to limited space, some of the missing proofs can only be found in the full version of the paper.

2 Preliminaries

Let $G_0 = (V, E_0)$ be the input directed multigraph. Let $n = |V|$ and $m = |E_0|$. Define $G = (V, E)$ to be a multigraph such that $E = E_0 \cup E_0^R$, $E_0 \cap E_0^R = \emptyset$, where E_0^R is the set of *reverse edges*. For any $uv = e \in E$, there is an edge $e^R \in E$ such that $e^R = vu$ and $(e^R)^R = e$. We have $e \in E_0$ iff $e^R \in E_0^R$.

Let $u : E_0 \rightarrow \mathbb{R}_+$ be a *capacity function*. A *flow* is a function $f : E \rightarrow \mathbb{R}$ such that for any $e \in E$ $f(e) = -f(e^R)$ and for each $e \in E_0$, $0 \leq f(e) \leq u(e)$. These conditions imply that for $e \in E_0$, $-u(e) \leq f(e^R) \leq 0$. We extend the function u to E by setting $u(e^R) = 0$ for all $e \in E_0$. Then, for all edges $e \in E$ we have $-u(e^R) \leq f(e) \leq u(e)$. The *unit capacity* function satisfies $u(e) = 1$ for all $e \in E_0$.

The *excess* $\text{exc}_f(v)$ of a vertex $v \in V$ is defined as $\sum_{uv=e \in E} f(e)$. Due to anti-symmetry of f , $\text{exc}_f(v)$ is equal to the amount of flow going into v by the edges of E_0 minus the amount of flow going out of v by the edges of E_0 . The vertex $v \in V$ is called an *excess* vertex if $\text{exc}_f(v) > 0$ and *deficit* if $\text{exc}_f(v) < 0$. Let X be the set of excess vertices of G and let D be the set of deficit vertices. Define the *total excess* Ψ_f as the sum of excesses of the excess vertices, i.e., $\Psi_f = \sum_{v \in X} \text{exc}_f(v) = \sum_{v \in D} -\text{exc}_f(v)$.

A flow f is called a *circulation* if there are no excess vertices, or equivalently, $\Psi_f = 0$.

Let $c : E_0 \rightarrow \mathbb{Z}$ be the input *cost* function. We extend c to E by setting $c(e^R) = -c(e)$ for all $e \in E_0$. The *cost* $c(f)$ of a flow f is defined as $\frac{1}{2} \sum_{e \in E} f(e)c(e) = \sum_{e \in E_0} f(e)c(e)$.

To *send a unit of flow* through $e \in E$ means to increase $f(e)$ by 1 and simultaneously decrease $f(e^R)$ by 1. By sending a unit of flow through e we increase the cost of flow by $c(e)$. To *send a unit of flow through a path* P means to send a unit of flow through each edge of P . In this case we also say that we *augment flow* f *along path* P .

The *residual network* G_f of f is defined as (V, E_f) , where $E_f = \{e \in E : f(e) < u(e)\}$.

Price functions and distances. We call any function $p : V \rightarrow \mathbb{R}$ a *price function* on G . The *reduced cost* of an edge $uv = e \in E$ wrt. p is defined as $c_p(e) := c(e) - p(u) + p(v)$. We call p a *feasible price function* of G if each edge $e \in E$ has nonnegative reduced cost wrt. p .

It is known that G has no negative-cost cycles (negative cycles, in short) if and only if some feasible price function p for G exists. If G has no negative cycles, distances in G (where we interpret c as a *length* function) are well-defined. For $u, v \in V$, we denote by $\delta_G(u, v)$ the distance between u and v , or, in other words, the length of a shortest $u \rightarrow v$ path in G .

► **Fact 1.** *Suppose G has no negative cycles. Let $t \in V$ be reachable in G from all vertices $v \in V$. Then the distance to function $\delta_{G,t}(v) := \delta_G(v, t)$ is a feasible price function of G .*

For $A, B \subseteq V(G)$ we sometimes write $\delta_G(A, B)$ to denote $\min\{\delta_G(u, v) : u \in A, v \in B\}$.

Planar graph toolbox. An r -*division* of a simple undirected plane graph G is a collection of $O(n/r)$ edge-induced subgraphs of G , called *pieces*, whose union is G and such that each piece P has $O(r)$ vertices and $O(\sqrt{r})$ *boundary vertices*. The boundary vertices ∂P of a piece P are the vertices of P shared with some other piece.

An r -*division with few holes* has an additional property that for each piece P , (1) P is connected, (2) there exist $O(1)$ faces of P whose union of vertex sets contains ∂P .

Let G_1, \dots, G_λ be some collection of plane graphs, where each G_i has a distinguished boundary set ∂G_i lying on $O(1)$ faces of G_i . A *distance clique* $DC(G_i)$ of G_i is defined as a complete digraph on ∂G_i such that the cost of the edge uv in $DC(G_i)$ is equal to $\delta_{G_i}(u, v)$.

► **Theorem 2** (MSSP [6, 25]). *Suppose a feasible price function on G_i is given. Then the distance clique $DC(G_i)$ can be computed in $O((|V(G_i)| + |E(G_i)| + |\partial G_i|^2) \log |V(G_i)|)$ time.*

The graph $DDG = DC(G_1) \cup \dots \cup DC(G_\lambda)$ is called a *dense distance graph*⁴.

► **Theorem 3** (FR-Dijkstra [13, 16]). *Given a feasible price function of DDG , single-source shortest paths in DDG can be computed in $O\left(\sum_{i=1}^\lambda |\partial G_i| \frac{\log^2 n}{\log^2 \log n}\right)$ time, where $n = |V(DDG)|$.*

Scaling framework for minimum-cost circulation. The following fact characterizes minimum circulations.

► **Fact 4** ([32]). *Let f be a circulation. Then $c(f)$ is minimum iff G_f has no negative cycles.*

It follows that a circulation f is minimum if there exists a feasible price function of G_f .

► **Definition 5** ([2, 18, 34]). *A flow f is ϵ -optimal wrt. price function p if for any $uv = e \in E_f$, $c(e) - p(u) + p(v) \geq -\epsilon$.*

The above notion of ϵ -optimality allows us, in a sense, to measure the optimality of a circulation: the smaller ϵ , the closer to the optimum a circulation f is. Moreover, if we deal with integral costs, $\frac{1}{n+1}$ -optimality is equivalent to optimality.

► **Lemma 6** ([2, 18]). *Suppose the cost function has integral values. Let circulation f be $\frac{1}{n+1}$ -optimal wrt. some price function p . Then f is a minimum cost circulation.*

⁴ Dense distance graphs have been defined differently multiple times in the literature. We use the definition of [16, 30] that captures all the known use cases (see [16] for discussion).

Proof. Suppose f is not minimum-cost. By Fact 4, f is not minimum-cost iff G_f contains a simple negative cycle C . Note that the cost of C is the same with respect to the cost functions c and c_p , as the prices cancel out. Therefore $\sum_{e \in C} c_p(e) \geq -\frac{n}{n+1} > -1$. But the cost of this cycle is integral and hence is at least 0, a contradiction. \blacktriangleleft

Let $C = \max_{e \in E_0} \{|c(e)|\}$. Suppose we have a procedure $\text{REFINE}(G, f_0, p_0, \epsilon)$ that, given a circulation f_0 in G that is 2ϵ -optimal wrt. p_0 , computes a pair (f', p') such that f' is a circulation in G , and it is ϵ -optimal wrt. p' . We use the general *scaling* framework, due to Goldberg and Tarjan [18], as given in Algorithm 1. By Lemma 6, it computes a min-cost circulation in G in $O(\log(nC))$ iterations. Therefore, if we implement REFINE to run in $T(n, m)$ time, we can compute a minimum cost circulation in G in $O(T(n, m) \log(nC))$ time.

■ **Algorithm 1** Scaling framework for min-cost circulation.

```

1: procedure MINIMUMCOSTCIRCULATION( $G$ )
2:    $f(e) := 0$  for all  $e \in G$ 
3:    $p(v) := 0$  for all  $v \in V$ 
4:    $\epsilon := C/2$ 
5:   while  $\epsilon > \frac{1}{n+1}$  do  $\triangleright f$  is  $2\epsilon$ -optimal wrt.  $p$ 
6:      $(f, p) := \text{REFINE}(G, f, p, \epsilon)$ 
7:      $\epsilon := \epsilon/2$ 
8:   return  $f$   $\triangleright f$  is circulation  $\frac{1}{n+1}$ -optimal wrt.  $p$ , i.e., a minimum-cost circulation

```

3 Refinement via Successive Approximate Shortest Paths

In this section we introduce our implementation of $\text{REFINE}(G, f_0, p_0, \epsilon)$. For simplicity, we start by setting $c(e) := c(e) - p_0(u) + p_0(v)$. After we are done, i.e., we have a circulation f' that is ϵ -optimal wrt. p' , (assuming costs reduced with p_0), we will return $(f', p' + p_0)$ instead. Therefore, we now have $c(e) \geq -2\epsilon$ for all $e \in E_{f_0}$.

Let f_1 be the flow initially obtained from f_0 by sending a unit of flow through each edge $e \in E_{f_0}$ such that $c(e) < 0$. Note that f_1 is ϵ -optimal, but it need not be a circulation.

We denote by f the *current flow* which we will gradually change into a circulation. Recall that X is the set of excess vertices of G and D is the set of deficit vertices (with respect to the current flow f). Recall a well-known method of finding the min-cost circulation exactly [5, 20, 23]: repeatedly send flow through shortest $X \rightarrow D$ paths in G_f . The sets X and D would only shrink in time. However, doing this on G_f exactly would be too costly. Instead, we will gradually convert f into a circulation, by sending flow from vertices of X to vertices of D but only using approximately (in a sense) shortest paths.

Let $\text{round}(y, z)$ denote the smallest integer multiple of z that is greater than y .

For any $e \in E$, set $c'(e) = \text{round}(c(e) + \epsilon/2, \epsilon/2)$. We define G'_f to be the “approximate” graph G_f with the costs given by c' instead of c .

For convenience, let us also define an extended version G''_f of G'_f to be G'_f with two additional vertices s (a super-excess-vertex) and t (a super-deficit-vertex) added. Let $M = \sum_{e \in E} |c'(e)| + \epsilon$. We also add to G''_f the following auxiliary edges:

1. an edge vt for all $v \in V$, we set $c'(vt) = 0$ if $v \in D$ and $c'(vt) = M$ otherwise,
2. an edge sx with $c'(sx) = 0$ for all $x \in X$.

Clearly, $\delta_{G''_f}(s, t) = \delta_{G'_f}(X, D)$ and every vertex in G''_f can reach t .

Our algorithm can be summarized very briefly, as follows. Start with $f = f_1$. While $X \neq \emptyset$, send a unit of flow along any shortest path P from X to D in G'_f (equivalently: from s to t in G''_f). Once finished, return f and $\delta_{G''_f,t}$ as the price function. The correctness of this approach follows from the following two facts that we discuss later on:

1. G'_f is negative-cycle free at all times,
2. after the algorithm finishes, f is a circulation in G that is ϵ -optimal wrt. $\delta_{G''_f,t}$.

If implemented naively, the algorithm would need $O(m)$ negative-weight shortest paths computations to finish. If we used Bellman-Ford method for computing shortest paths, the algorithm would run in $O(nm^2)$ time. To speed it up, we apply two optimizations.

First, as in the successive shortest paths algorithm for general graphs [10, 35], we observe that the distances $\delta_{G''_f,t}$ computed before sending flow through a found shortest $s \rightarrow t$ path constitute a feasible price function of G''_f after augmenting the flow. This allows us to replace Bellman-Ford algorithm with Dijkstra's algorithm and reduce the time to $O(m^2 + nm \log n)$. Next, instead of augmenting the flow along a single shortest $X \rightarrow D$ path, we send flow through a maximal set of edge-disjoint shortest $X \rightarrow D$ paths, as in Hopcroft-Karp algorithm for maximum bipartite matching [19]. Such a set can be easily found in $O(m)$ time when the distances to t in G''_f are known. This way, we finish after only $O(\sqrt{m})$ phases of shortest path computation and flow augmentation. The pseudocode is given in Algorithm 2.

■ **Algorithm 2** Refinement via successive shortest paths.

Require: f_0 is a circulation in G 2ϵ -feasible wrt. p_0

Require: $\text{DISTANCETO}(H, t, p)$ computes the vector of distances (i.e., $\delta_{G,t}$) from all $v \in V(H)$ to $t \in V(H)$, where p is a feasible price function of H .

Require: $\text{SENDFLOW}(f, E^*)$ returns a flow f' such that $f'(e)$ equals $f(e) + 1$ if $e \in E^*$, $f(e) - 1$ if $e^R \in E^*$, and $f(e)$ otherwise.

Output: (f, p) , where f is a circulation in G ϵ -feasible wrt. p

```

1: procedure REFINE( $G, f_0, p_0, \epsilon$ )
2:    $c(e) := c(e) - p_0(u) + p_0(v)$  for all  $e = uv \in E$ .
3:    $f := \text{SENDFLOW}(f_0, \{e \in E_{f_0} : c(e) < 0\})$ 
4:    $p(v) := 0$  for all  $v \in V$ 
5:   while  $X \neq \emptyset$  do                                      $\triangleright p$  is a feasible price function of  $G''_f$ 
6:     Construct  $G''_f$  out of  $G'_f$ .
7:      $p := \text{DISTANCETO}(G''_f, t, p)$ 
8:      $Q_0, \dots, Q_k :=$  a maximal set of edge-disjoint  $s \rightarrow t$  paths in  $G''_f$  consisting solely
           of edges satisfying  $c'_p(e) = 0$ .
9:      $f := \text{SENDFLOW}(f, E((Q_0 \cup \dots \cup Q_k) \cap G'_f))$ 
10:  return  $(f, \text{DISTANCETO}(G''_f, t, p) + p_0)$               $\triangleright f$  is  $\epsilon$ -feasible wrt.  $\delta_{G''_f,t} + p_0$ 

```

3.1 Analysis

Below we state some key properties of our refinement method. The proofs are can be found in the full version of the paper.

► **Lemma 7.** *Suppose G''_f has no negative cycles. Then f is ϵ -optimal wrt. $\delta_{G''_f,t}$.*

Proof. Recall that G_f and G'_f have the same sets of edges, only different costs. Let $uv = e \in G_f$. Set $p := \delta_{G''_f,t}$. By Fact 1, $c'(e) - p(u) + p(v) \geq 0$. Note that $c(e) \geq c'(e) - \epsilon$. Hence, $c(e) - p(u) + p(v) \geq c'(e) - p(u) + p(v) - \epsilon \geq -\epsilon$. ◀

► **Lemma 8.** *If $X \neq \emptyset$, then there exists a path from X to D in G_f .*

Before we proceed further, we need to introduce more notation. Let Δ denote the length of the shortest $X \rightarrow D$ path in G'_f (Δ changes in time along with f).

Let $q = \Psi_{f_1}$. Clearly, $q \leq m$. For $i = 1, \dots, q$, denote by f_{i+1} the flow (with total excess $q - i$) obtained from f_i by sending a unit of flow through an arbitrarily chosen shortest $X \rightarrow D$ path P_i of G'_{f_i} .

For $i = 1, \dots, q$, let Δ_i be the value Δ when $f = f_i$. We set $\Delta_{q+1} = \infty$.

► **Lemma 9.** *Let $p_i^* : V \cup \{s, t\} \rightarrow \{k \cdot \epsilon/2 : k \in \mathbb{Z}\}$ be defined as $p_i^* = \delta_{G'_{f_i}, t}$. Then:*

1. G'_{f_i} has no cycles of non-positive cost,
2. for any $e \in P_i$, the reduced cost of e^R wrt. p_i^* is positive,
3. p_i^* is a feasible price function of both G'_{f_i} and $G''_{f_{i+1}}$,
4. $0 < \Delta_i \leq \Delta_{i+1}$.

By Lemmas 8 and 9, our general algorithm computes a circulation f_{q+1} such that p_q^* is a feasible price function of $G'_{f_{q+1}}$. Since f_{q+1} has no negative cycles, by Lemma 7, f_{q+1} is ϵ -optimal wrt. $\delta_{G'_{f_{q+1}}, t}$. We conclude that the algorithm is correct.

The following lemma is the key to the running time analysis.

► **Lemma 10.** *If $X \neq \emptyset$ (equivalently, if $\Delta < \infty$), then $\Psi_f \cdot \Delta \leq 6\epsilon m$.*

3.2 Efficient Implementation

As mentioned before, we could use Lemma 9 directly: start with flow f_1 and $p_0^* \equiv 0$. Then, repeatedly compute a shortest $X \rightarrow D$ path P_i along with the values p_i^* using Dijkstra's algorithm on G'_f (with the help of price function p_{i-1}^* to make the edge costs non-negative), and send flow through P_i to obtain f_{i+1} . However, we can also proceed as in Hopcroft-Karp algorithm and augment along many shortest $X \rightarrow D$ paths of cost Δ at once. We use the following lemma.

► **Lemma 11.** *Let p be a feasible price function of G'_f . Suppose there is no $s \rightarrow t$ path in G'_f consisting of edges with reduced (wrt. p) cost 0. Then $\Delta = \delta_{G'_f}(X, D) > p(s) - p(t)$.*

Suppose we run the simple-minded algorithm. Assume that at some point $f = f_i$, and we have p_i^* computed. Any $s \rightarrow t$ path in G''_{f_i} with reduced (wrt. p_i^*) cost 0 corresponds to some shortest $X \rightarrow D$ path (of length Δ_i) in G'_f . Additionally, we have $p_i^*(s) = 0$ and $p_i^*(t) = \Delta_i$.

Let Q_0, \dots, Q_k be some maximal set of edge-disjoint $s \rightarrow t$ paths in G''_{f_i} with reduced cost 0. By Lemma 9, we could in principle choose $P_i = Q_0, P_{i+1} = Q_1, \dots, P_{i+k} = Q_k$ and this would not violate the rule that we repeatedly choose shortest $X \rightarrow D$ paths.

Moreover, p_i^* is a feasible price function of $G''_{f_{i+1}}$ for any choice of $P_i = Q_j, j = 0, \dots, k$. Hence, the reduced cost wrt. p_i^* of any $e^R \in Q_j$, is non-negative. Therefore, in fact p_i^* is a feasible price function of all $G''_{f_{i+1}}, G''_{f_{i+2}}, \dots, G''_{f_{i+k+1}}$. On the other hand, since for all $e \in P_i \cup \dots \cup P_{i+k}$, the reduced cost (wrt. p_i^*) of e^R is positive, and the set Q_0, \dots, Q_k was maximal, we conclude that there is no $s \rightarrow t$ path in $G''_{f_{i+k+1}}$ consisting only of edges with reduced cost (wrt. p_i^*) 0. But $p_i^*(s) - p_i^*(t) = \Delta_i$, so by Lemma 11 we have $\Delta_{i+k+1} > \Delta_i$.

Since we can choose a maximal set Q_0, \dots, Q_k using a DFS-style procedure in $O(m)$ time (for details, see Section 4.3, where we take a closer look at it to implement it faster in the planar case), we can actually move from f_i to f_{i+k+1} and simultaneously increase Δ in $O(m)$ time. Since p_i^* is a feasible price function of $G''_{f_{i+k+1}}$, the new price function p_{i+k+1}^* can be computed, again, using Dijkstra's algorithm. The total running time of this algorithm is $O(m + n \log n)$ times the number of times Δ increases.

► **Lemma 12.** *The value Δ changes $O(\sqrt{m})$ times.*

Proof. By Lemma 9, Δ can only increase, and if it does, it increases by at least $\epsilon/2$. After it increases $2\sqrt{m}$ times, $\Delta \geq \epsilon\sqrt{m}$. But then, by Lemma 10, Ψ_f is no more than $6\sqrt{m}$. As each change of Δ is accompanied with some decrease of Ψ_f , Δ can change $O(\sqrt{m})$ times more. ◀

► **Theorem 13.** *REFINE as implemented in Algorithm 2 runs in $O((m + n \log n)\sqrt{m})$.*

We can in fact improve the running time to $O(m\sqrt{m})$ by taking advantage of so-called Dial's implementation of Dijkstra's algorithm [9]. The details are deferred to the full version.

3.3 Bounding the Total Length of Augmenting Paths

► **Fact 14.** *For every $e \in E$ we have $c'(e) + c'(e^R) > \epsilon$.*

Proof. We have $c'(e) > c(e) + \epsilon/2$. Hence, $c'(e) + c'(e^R) > c(e) + \epsilon/2 + c(e^R) + \epsilon/2 = \epsilon$. ◀

There is a subtle reason why we set $c'(e)$ to be $\text{round}(c(e) + \epsilon/2, \epsilon/2)$ instead of $\text{round}(c(e), \epsilon)$. Namely, this allows us to obtain the following bound.

► **Lemma 15.** *For any $i = 1, \dots, q$ we have $c'(f_{i+1}) - c'(f_i) < \Delta_i - \frac{1}{2}|P_i| \cdot \epsilon$.*

Proof. We have

$$c'(f_{i+1}) - c'(f_i) = \frac{1}{2} \sum_{e \in E} (f_{i+1}(e) - f_i(e))c'(e) = \frac{1}{2} \sum_{e \in P_i} (c'(e) - c'(e^R)).$$

By Fact 14, $-c'(e^R) < c'(e) - \epsilon$ for all $e \in E$. Hence

$$c'(f_{i+1}) - c'(f_i) < \sum_{e \in P_i} c'(e) - \frac{1}{2}|P_i| \cdot \epsilon = \Delta_i - \frac{1}{2}|P_i| \cdot \epsilon. \quad \blacktriangleleft$$

► **Lemma 16.** *Let f^* be any flow. Then $c(f_0) - c(f^*) \leq 2\epsilon m$.*

Proof. We have

$$c(f_0) - c(f^*) = \frac{1}{2} \sum_{e \in E} (f_0(e) - f^*(e))c(e).$$

If $f_0(e) > f^*(e)$, then $e^R \in E_{f_0}$ and hence $c(e^R) \geq -2\epsilon$, and thus $c(e) \leq 2\epsilon$. Otherwise, if $f_0(e) < f^*(e)$ then $e \in E_{f_0}$ and $c(e) \geq -2\epsilon$.

In both cases $(f_0(e) - f^*(e))c(e) \leq 2\epsilon$. Therefore, since $|E| = 2m$, $c(f_0) - c(f^*) \leq 2\epsilon m$. ◀

► **Lemma 17.** *Let f^* be any flow. Then $|c'(f^*) - c(f^*)| \leq \epsilon m$.*

Proof. Recall that we had $0 < c'(e) - c(e) \leq \epsilon$. Hence $|f^*(e)(c'(e) - c(e))| \leq \epsilon$ and:

$$|c'(f^*) - c(f^*)| = \frac{1}{2} \left| \sum_{e \in E} f^*(e)(c'(e) - c(e)) \right| \leq \frac{1}{2} \sum_{e \in E} |f^*(e)(c'(e) - c(e))| \leq \frac{1}{2} \sum_{e \in E} \epsilon = \epsilon m. \quad \blacktriangleleft$$

The inequalities from Lemmas 15, 16 and 17 combined give us the following important property of the set of paths we augment along.

► **Lemma 18.** *The total number of edges on all the paths we send flow through is $O(m \log m)$.*

Proof. By Lemma 16 and the fact that $c(f_1) \leq c(f_0)$, we have:

$$c(f_1) - c(f_{q+1}) \leq c(f_0) - c(f_{q+1}) \leq 2\epsilon m.$$

On the other hand, by Lemma 17 and Lemma 15, we obtain:

$$\begin{aligned} c(f_1) - c(f_{q+1}) &\geq (c'(f_1) - \epsilon m) + (-c'(f_{q+1}) - \epsilon m) = c'(f_1) - c'(f_{q+1}) - 2\epsilon m \\ &= \sum_{i=1}^q (c'(f_i) - c'(f_{i+1})) - 2\epsilon m \geq \sum_{i=1}^q \left(\frac{1}{2}|P_i| \cdot \epsilon - \Delta_i\right) - 2\epsilon m. \end{aligned}$$

By combining the two inequalities and applying Lemma 10, we get:

$$\sum_{i=1}^q \frac{1}{2}|P_i| \leq 4m + \sum_{i=1}^q \frac{\Delta_i}{\epsilon} \leq 4m + \sum_{i=1}^q \frac{6m}{\Psi_{f_i}} = 4m + \sum_{i=1}^q \frac{6m}{q-i+1} = O(m \log m). \quad \blacktriangleleft$$

4 Unit-Capacity Min-Cost Circulation in Planar Graphs

In this section we show that the refinement algorithm per scale from Section 3 can be simulated on a planar digraph more efficiently. Specifically, we prove the following theorem.

► **Theorem 19.** *REFINE can be implemented on a planar graph in $\tilde{O}((nm)^{2/3})$ time.*

Let $r \in [1, n]$ be a parameter. Suppose we are given an r -division with few holes $\mathcal{P}_1, \dots, \mathcal{P}_\lambda$ of G such that for any i we have $\lambda = O(n/r)$, $|V(\mathcal{P}_i)| = O(r)$, $|\partial\mathcal{P}_i| = O(\sqrt{r})$, $\partial\mathcal{P}_i$ lies on $O(1)$ faces of \mathcal{P}_i , and the pieces are edge-disjoint. We set $\partial G = \bigcup_{i=1}^\lambda \partial\mathcal{P}_i$. Clearly, $|\partial G| = O(n/\sqrt{r})$.

In the full version we show that we can reduce our instance to the case when the above assumptions are satisfied in nearly-linear time.

Since m might be $\omega(n)$, we cannot really guarantee that $|E(\mathcal{P}_i)| = O(r)$. This will not be a problem though, since, as we will see, for all the computations involving the edges of \mathcal{P}_i (e.g., computing shortest paths in \mathcal{P}_i , or sending a unit of flow through a path of \mathcal{P}_i) of all edges $uv = e \in E(\mathcal{P}_i)$ we will only care about an edge $e \in E(\mathcal{P}_i) \cap G_f$ with minimal cost $c'(e)$. Therefore, since \mathcal{P}_i is planar, at any time only $O(r)$ edges of \mathcal{P}_i will be needed.

Recall that the per-scale algorithm for general graphs (Algorithm 2) performed $O(\sqrt{m})$ phases, each consisting of two steps: a shortest path computation (to compute the price function p^* from Lemma 9), followed by the computation of a maximal set of edge-disjoint augmenting paths of reduced (wrt. p^*) cost 0. We will show how to implement both steps in $\tilde{O}(n/\sqrt{r})$ amortized time, at the additional total data structure maintenance cost (over all phases) of $\tilde{O}(mr)$. Since there are $O(\sqrt{m})$ steps, this will yield $\tilde{O}(nm)^{2/3}$ time by appropriately setting r .

We can maintain the flow f explicitly, since it undergoes only $O(m \log n)$ edge updates (by Lemma 18). However, we will not compute the entire price function p^* at all times explicitly, as this is too costly. Instead, we will only compute p^* limited to the subset $\partial G \cup \{s, t\}$.

For each \mathcal{P}_i , define $\mathcal{P}'_{f,i} = G'_f \cap \mathcal{P}_i$. We also define $\mathcal{P}''_{f,i}$ to be $\mathcal{P}'_{f,i}$ with vertices $\{s, t\}$ added, and those edges sv, vt of G''_f that satisfy $v \in V(\mathcal{P}_i) \setminus \partial\mathcal{P}_i$. This way, $\mathcal{P}''_{f,i} \subseteq G''_f$ and $E(\mathcal{P}''_{f,i}) \cap E(\mathcal{P}''_{f,j}) = \emptyset$ for $i \neq j$. The costs of edges $e \in E(\mathcal{P}''_{f,i})$ are the same as in G''_f , i.e., $c'(e)$. Besides, for each i we will store a “local” price function p_i that is feasible only for $\mathcal{P}''_{f,i}$.

After the algorithm finishes, we will know how the circulation looks like precisely. However, the general scaling algorithm requires us to also output price function p such that f is an ϵ -optimal circulation wrt. p . f is ϵ -optimal wrt. p^* in the end, but we will only have it computed for the vertices $\partial G \cup \{s, t\}$. Therefore, we extend it to all remaining vertices of G .

► **Lemma 20.** *Suppose we are given the values of p^* on $\partial\mathcal{P}_i$ and a price function p_i feasible for $\mathcal{P}_{f,i}''$. Then we can compute the values $p^*(u)$ for all $v \in V(\mathcal{P}_{f,i}'')$ in $O(r \log r)$ time.*

Hence, in order to extend p^* to all vertices of G once the final circulation is found, we apply Lemma 20 to all pieces. This takes $O(\frac{n}{r} \cdot r \log r) = O(n \log n)$ time.

4.1 Dijkstra Step

Let us start with an implementation of the Dijkstra step computing the new price function p^* . First, for each piece \mathcal{P}_i we define the compressed version $H_{f,i}''$ of $\mathcal{P}_{f,i}''$ as follows. Let $V(H_{f,i}'') = \partial\mathcal{P}_i \cup \{s, t\}$. The set of edges of $H_{f,i}''$ is formed by:

- a distance clique $\text{DC}(\mathcal{P}_{f,i}'')$ between vertices $\partial\mathcal{P}_i$ in $\mathcal{P}_{f,i}''$,
- for each $v \in \partial\mathcal{P}_i$, an edge sv of cost $\delta_{\mathcal{P}_{f,i}''}(s, v)$ if this distance is finite,
- for each $v \in \partial\mathcal{P}_i$, an edge vt of cost $\delta_{\mathcal{P}_{f,i}''}(v, t)$ if this distance is finite,
- an edge st of cost $\delta_{\mathcal{P}_{f,i}''}(s, t)$ if this distance is finite.

Recall that we store a price function p_i of $\mathcal{P}_{f,i}''$. Therefore, by Theorem 2, $\text{DC}(\mathcal{P}_{f,i}'')$ can be computed in $O(r \log r)$ time. All needed distances $\delta_{\mathcal{P}_{f,i}''}(s, v)$ and $\delta_{\mathcal{P}_{f,i}''}(v, t)$ can be computed in $O(r \log r)$ time using Dijkstra's algorithm (again, with the help of price function p_i).

Now define H_f'' to be $\bigcup_{i=1}^{\lambda} H_{f,i}''$ with edges sv and vt of G_f'' that satisfy $v \in \partial G$ added.

► **Fact 21.** *For any $u, v \in V(H_f'')$, $\delta_{H_f''}(u, v) = \delta_{G_f''}(u, v)$.*

Observe that H_f'' is a dense distance graph in terms of the definition of Section 2: it consists of $O(n/r)$ distance cliques $\text{DC}(\mathcal{P}_{f,i}'')$ with $O(\sqrt{r})$ vertices each, and $O(n/\sqrt{r})$ additional edges which also can be interpreted as 2-vertex distance cliques.

Hence, given a feasible price function on H_f'' , we can compute distances to t in H_f'' on it using Theorem 3 in $O\left(n/\sqrt{r} \cdot \frac{\log^2 n}{\log^2 \log n}\right)$ time. Since $V(H_f'') = \partial G \cup \{s, t\}$, the price function p^* we have is indeed sufficient. The computed distances to t form the new price function p^* on $\partial G \cup \{s, t\}$ as in the algorithm for general graphs (see Algorithm 2).

4.2 Sending Flow Through a Path

In the general case updating the flow after an augmenting path has been found was trivial. However, as we operate on a compressed graph, the update procedure has to be more involved.

Generally speaking, we will repeatedly find some shortest $s \rightarrow t$ path $Q = e_1 \dots e_k$ in H_f'' , translate it to a shortest $s \rightarrow t$ path \mathcal{P} in G_f'' and send flow through it. It is easy to see by the definition of H_f'' that Q can be translated to a shortest $s \rightarrow t$ path in G_f'' and vice versa. Each edge e_j can be translated to either some subpath inside a single graph $\mathcal{P}_{f,i}''$, or an edge of G_f'' of the form sv or vt , where $v \in \partial G$. This can be done in $O(r \log n)$ time by running Dijkstra's algorithm on $\mathcal{P}_{f,i}''$ with price function p_i . We will guarantee that path P obtained by concatenating the translations of individual edges e_j contains no repeated edges of G_f'' .

We now show how to update each $H_{f,i}''$ after sending flow through the found path P . Note that we only need to update $H_{f,i}''$ if $E(P) \cap E(\mathcal{P}_{f,i}'') \neq \emptyset$. In such case we call \mathcal{P}_i an *affected piece*. Observe that some piece can be affected at most $O(m \log m)$ times since the total number of edges on all shortest augmenting paths P in the entire algorithm, regardless of their choice, is $O(m \log m)$ (see Lemma 18).

To rebuild $H_{f,i}''$ to take into account the flow augmentation we will need a feasible price function on $\mathcal{P}_{f,i}''$ after the augmentation. However, we cannot be sure that what we have, i.e., p_i , will remain a good price function of $\mathcal{P}_{f,i}''$ after the augmentation. By Lemma 9, luckily, we know that p^* is a feasible price function after the augmentation for the whole

graph G_f'' . In particular, p^* (before the augmentation) limited to $V(\mathcal{P}_{f,i}'')$ is a feasible price function of $\mathcal{P}_{f,i}''$ after the augmentation. Hence, we can compute new p_i equal to p^* using Lemma 20 in $O(r \log r)$ time. Given a feasible price function p_i on $\mathcal{P}_{f,i}''$ after f is augmented, we can recompute $H_{f,i}''$ in $O(r \log r)$ time as discussed in Section 4.1. We conclude that the total time needed to update the graph H_f'' subject to flow augmentations is $O(mr \log r \log m) = O(mr \log n \log m)$.

4.3 A Path Removal Algorithm

In this section we consider an abstract “path removal” problem, that generalizes the problem of finding a maximal set of edge-disjoint $s \rightarrow t$ paths. We will use it to reduce the problem of finding such a set of paths on a subgraph of G_f'' consisting of edges with reduced cost 0 wrt. p^* to the problem of finding such a set of paths on the zero-reduced cost subgraph of H_f'' .

Suppose we have some directed acyclic graph H with a fixed source s and sink t , that additionally undergoes some limited adversarial changes. We are asked to efficiently support a number of *rounds*, until t ceases to be reachable from s . Each round goes as follows.

1. We first find either any $s \rightarrow t$ path P , or detect that no $s \rightarrow t$ path exists.
2. Let $E^+ \subseteq V \times V$, and $P \subseteq E^- \subseteq E(H)$ be some adversarial sets of edges. Let $H' = (V, E')$, where $E' = E(H) \setminus E^- \cup E^+$. Assume that for any $v \in V(H)$, if v cannot reach t in H , then v cannot reach t in H' either. Then the adversarial change is to remove E^- from E and add E^+ to E , i.e., set $E(H) = E'$.

Let $\bar{n} = |V(H)|$ and let \bar{m} be the number of edges ever seen by the algorithm, i.e., the sum of $|E(H)|$ and all $|E^+|$. We will show an algorithm that finds all the paths P in $O(\bar{n} + \bar{m})$ total time. Let us also denote by $\bar{\ell}$ the sum of lengths of all returned paths P . Clearly, $\bar{\ell} \leq \bar{m}$.

A procedure handling the phase 1 of each round, i.e., finding a $s \rightarrow t$ path or detecting that there is none, is given in Algorithm 3. The second phase of each round simply modifies the representation of the graph H accordingly. Throughout all rounds, we store a set W of vertices w of H for which we have detected that there is no more $w \rightarrow t$ path in H . Initially, $W = \emptyset$. Each edge $e \in E(H)$ can be *scanned* or *unscanned*. Once e is scanned, it remains scanned forever. The adversarial edges E^+ that are inserted to $E(H)$ are initially unscanned. The following lemmas establishing the correctness and efficiency of the crucial parts of Algorithm 3 are all proved in the full version.

■ **Algorithm 3** Path-finding procedure. Returns a $s \rightarrow t$ path in H or detects there is none.

```

1: procedure FINDPATH( $H$ )
2:    $Q :=$  an empty path with a single endpoint  $s$                                 ▷  $Q$  is an  $s \rightarrow s$  path
3:   while  $s \notin W$  and the other endpoint  $y$  of  $Q$  is not equal to  $t$  do          ▷  $Q$  is an  $s \rightarrow y$  path
4:     if there exists an unscanned edge  $yv = e \in E(H)$  such that  $v \notin W$  then
5:       mark  $e$  scanned
6:        $Q := Qe$ 
7:     else
8:        $W := W \cup \{y\}$ 
9:       remove the last edge of  $Q$  unless  $Q$  is empty
10:  if  $Q = \emptyset$  then
11:    report  $t$  not reachable from  $s$  and stop
12:  else
13:    return  $Q$  and  $Q := 0$ .

```

► **Lemma 22.** *Algorithm 3 correctly finds an $s \rightarrow t$ path in H or detects there is none.*

► **Lemma 23.** *The total number of times line 9 is executed, through all rounds, is $O(\bar{n})$.*

► **Lemma 24.** *Line 6 of Algorithm 3 is executed $O(\bar{n} + \bar{\ell})$ times through all rounds.*

► **Lemma 25.** *The total time used by Algorithm 3, through all rounds, is $O(\bar{n} + \bar{m})$.*

4.4 Finding a Maximal Set of Shortest Augmenting Paths

Recall that for a general graph, after computing the price function p^* we found a maximal set of edge-disjoint $s \rightarrow t$ paths in the graph Z_f'' , defined as a subgraph of G_f'' consisting of edges with reduced cost 0 (wrt. p^*). To accomplish that, we could in fact use the path removal algorithm from Section 4.3 run on Z_f'' : until there was an $s \rightarrow t$ path in Z_f'' , we would find such a path P , remove edges of P (i.e., set $E^- = P$ and $E^+ = \emptyset$), and repeat. Since in this case we never add edges, the assumption that t cannot become reachable from any v due to updating Z_f'' is met.

Let Y_f'' be the subgraph of the graph H_f'' from Section 4.1 consisting of edges with reduced (wrt. p^*) cost 0. Since all edges of H_f'' correspond to shortest paths in G_f'' , all edges of Y_f'' correspond to paths in G_f'' with reduced cost 0. Because Z_f'' is acyclic by Lemma 9, Y_f'' is acyclic as well. Moreover, for any two edges $e_1, e_2 \in E(Y_f'')$, if there is a path going through both e_1 and e_2 in Y_f'' , then the paths represented by e_1 and e_2 are edge-disjoint in Z_f'' (as otherwise Z_f'' would have a cycle). Therefore, any path Q in Y_f'' translates to a simple path in $Z_f'' \subseteq G_f''$.

We will now explain why running Algorithm 3 on Y_f'' can be used to find a maximal set of edge-disjoint $s \rightarrow t$ paths. Indeed, by Fact 21, Y_f'' contains an $s \rightarrow t$ path iff Z_f'' does. Since Y_f'' is just a compressed version of Z_f'' , and Z_f'' undergoes edge deletions only (since we only remove the found paths), the updates to Y_f'' cannot make some t reachable from some new vertex $v \in V(Y_f'')$. Technically speaking, we should think of Y_f'' as undergoing both edge insertions and deletions: whenever some path $Q \subseteq Y_f''$ is found, we include Q in E^- and send the flow through a path corresponding to Q in G_f'' , as described in Section 4.2. But then for all affected pieces \mathcal{P}_i , $H_{f,i}''$ is recomputed and thus some of the edges of Q might be reinserted to Y_f'' again. These edges should be seen as forming the set E^+ , whereas the old edges of the recomputed graphs $H_{f,i}''$ belong to E^- . In terms of the notation of Section 4.3, when running Algorithm 3 on Y_f'' , we have $\bar{n} = O(n/\sqrt{r})$. The sum of values $\bar{\ell}$ from Section 4.3 over all phases of the algorithm is, by Lemma 18, $O(m \log m)$. Similarly, again by Lemma 18, the sum of the values \bar{m} from Section 4.3 over all phases, is $O(m^{3/2} + mr^2 \log m)$ (since each time E^+ might be as large as r^2 times the number of affected pieces).

Recall that there are $O(\sqrt{m})$ phases, and the total time needed to maintain the graph H_f'' subject to flow augmentations is $O(mr \log r \log m)$ (see Section 4.2). For each phase, running a Dijkstra step to compute p^* using FR-Dijkstra, followed by running Algorithm 3 directly until there are no $s \rightarrow t$ paths in Y_f'' would lead to $O\left(\sqrt{m} \left(\frac{n}{\sqrt{r}} \frac{\log^2 n}{\log^2 \log n}\right) + m^{3/2} + mr^2 \log m\right)$ total time, i.e., would not yield any improvement over the general algorithm. However, we can do better by implementing Algorithm 3 on Y_f'' more efficiently.

► **Lemma 26** ([21, 24]). *Let Z be the subgraph of $\mathcal{P}'_{f,i}$ consisting of edges with reduced cost 0 with respect to some feasible price function p . There exists $O(\sqrt{r})$ pairs of subsets $(A_{i,1}, B_{i,1}), (A_{i,2}, B_{i,2}), \dots$ of $\partial\mathcal{P}_i$ such that for each $v \in \partial\mathcal{P}_i$:*

- *The number of sets $A_{i,j}$ ($B_{i,j}$) such that $v \in A_{i,j}$ ($v \in B_{i,j}$, resp.) is $O(\log r)$.*
- *Each $B_{i,j}$ is totally ordered according to some order $\prec_{i,j}$.*

- For any j such that $v \in A_{i,j}$, there exist $l_{i,v,j}, r_{i,v,j} \in B_{i,j}$ such that the subset $R_{i,v}$ of $\partial\mathcal{P}_i$ reachable from v in Z can be expressed as $\bigcup_{j:v \in A_{i,j}} \{w \in B_{i,j} : l_{i,v,j} \preceq_{i,j} w \preceq_{i,j} r_{i,v,j}\}$. The sets $A_{i,j}$, $B_{i,j}$ and the vertices $l_{i,v,j}, r_{i,v,j}$ for all v, j can be computed in $O(\sqrt{r} \log r)$ time based on the distance clique between $\partial\mathcal{P}_i$ in $\mathcal{P}'_{f,i}$ and the values of p^* on $\partial\mathcal{P}_i$.

Recall that in Section 4.3, to bound the total running time, it was enough to bound the total time spent on executing lines 4, 6 and 9. We will show that using Lemma 26, in terms of the notation from Section 4.3, we can make the total time spent on executing line 4 only $\tilde{O}(\bar{n} + \bar{\ell})$ instead of $O(\bar{m})$, at the cost of increasing the total time of executing line 9 to $\tilde{O}(\bar{n})$.

Specifically, at the beginning of each phase we compute the data from Lemma 26 for all pieces \mathcal{P}_i . Since for all i we have the distance cliques $\text{DC}(\mathcal{P}''_{f,i})$ computed, this takes $O(\frac{n}{r} \cdot \sqrt{r} \log r) = O(n/\sqrt{r} \log n)$ time. We will also recompute the information of Lemma 26 for an affected piece \mathcal{P}_i after $H''_{f,i}$ is recomputed. As the total number of times some piece is affected is $O(m \log m)$, this takes $O(m\sqrt{r} \log r \log m)$ time through all phases.

Whenever the data of Lemma 26 is computed for some piece \mathcal{P}_i , for each pair $(A_{i,j}, B_{i,j})$ we store $B_{i,j} \cap W$ in a dynamic predecessor/successor data structure $D_{i,j}$, sorted by $\prec_{i,j}$. For each $v \in \partial\mathcal{P}_i$ and j such that $v \in A_{i,j}$ we store a vertex $next_{i,v,j}$ initially equal to $l_{i,v,j}$. It is easy to see that these auxiliary data structures can be constructed in time linear in their size, i.e., $O(\sqrt{r} \log r)$ time. Hence, the total cost of computing them is $O(\sqrt{mn}/\sqrt{r} \log n + m\sqrt{r} \log r \log m) = O\left(\sqrt{m}\left(\frac{n}{\sqrt{r}} \frac{\log^2 n}{\log^2 \log n}\right) + mr \log n \log m\right)$.

Now, to implement line 9, when y is inserted into W we go through all pieces \mathcal{P}_i such that $y \in \partial\mathcal{P}_i$ and all $B_{i,j}$ such that $y \in B_{i,j}$. For each such (i, j) , we remove y from $D_{i,j}$ in $O(\log \log n)$ time. Recall that the sum of numbers of such pairs (i, j) over all $v \in \partial G$ is $O(\sum_{i=1}^{\lambda} |\partial\mathcal{P}_i| \log r) = O(n/\sqrt{r} \log n)$. Hence, by Lemma 23 the total time spent on executing line 9 in a single phase is $O(n/\sqrt{r} \log n \log \log n)$.

Finally, we implement line 4 as follows. The unscanned edges of Y_f'' that are not between boundary vertices are handled in a simple-minded way as in Lemma 25. There are only $O(n/\sqrt{r})$ of those, so we can neglect them. In order to be able to efficiently find some unscanned edge yv such that $y, v \in \partial G$ and $v \notin W$, we keep for any $v \in \partial G$ a set U_v of pieces \mathcal{P}_i such that $v \in \partial\mathcal{P}_i$ and there may still be some unscanned edges from v to $w \in \partial\mathcal{P}_i$ in $H''_{f,i}$. Similarly, for each $\mathcal{P}_i \in U_v$ we maintain a set $U_{v,i}$ of data structures $D_{i,j}$ such that $next_{i,v,j} \neq \mathbf{nil}$. Whenever the data of Lemma 26 is computed for \mathcal{P}_i , \mathcal{P}_i is inserted back to U_v for all $v \in \partial\mathcal{P}_i$, and the sets $U_{v,i}$ are recomputed with no additional asymptotic overhead. To find an unscanned edge yv , for each $\mathcal{P}_i \in U_y$ we proceed as follows. We attempt to find an unscanned edge yv in \mathcal{P}_i . If we succeed or U_y is empty, we stop. Otherwise we remove \mathcal{P}_i from U_y and repeat, i.e., try another $\mathcal{P}_j \in U_y$, unless U_y is empty. To find an unscanned edge yv from a piece \mathcal{P}_i , we similarly try to find an unscanned edge yv in subsequent data structures $D_{i,j} \in U_{v,i}$, and remove the data structures for which we fail from $U_{v,i}$. For a single data structure $D_{i,j}$, we maintain an invariant that an edge yw , $w \in D_{i,j}$ has been scanned iff $w \prec_{i,j} next_{i,v,j}$. Hence, to find the next unscanned edge, we first find $x \in D_{i,j}$ such that $next_{i,v,j} \preceq_{i,j} x$ and x is smallest possible. This can be done in $O(\log \log n)$ time since $D_{i,j}$ is a dynamic successor data structure. If x does not exist or $r_{i,v,j} \prec x$, then, by Lemma 26, there are no more unscanned edges yw such that $w \in D_{i,j}$, and thus we remove $D_{i,j}$ from $U_{v,i}$. Otherwise, we return an edge yx and set $next_{i,v,j}$ to be the successor of x in $D_{i,j}$ (or possibly $next_{i,v,j} := \mathbf{nil}$ if none exists), again in $O(\log \log n)$ time.

Observe that all “failed” attempts to find an edge yv , where $v \in \partial G$ can be charged to an insertion of some \mathcal{P}_i to U_y or to an insertion of some $D_{i,j}$ to $U_{v,i}$. The total number of such insertions is again $O\left(\sqrt{m} \frac{n}{\sqrt{r}} \log n + m\sqrt{r} \log r \log m\right)$. A successful at-

tempt, on the other hand, costs $O(\log \log n)$ worst-case time. Since line 4 is executed $O(\sqrt{mn}/\sqrt{r} + m \log n)$ times through all phases, the total time spent on executing line 4 is again $O\left(\sqrt{m}\left(\frac{n}{\sqrt{r}} \frac{\log^2 n}{\log n}\right) + mr \log n \log m\right)$. By setting $r = \frac{n^{2/3}}{m^{1/3}} \cdot \left(\frac{\log n}{\log m \cdot \log^2 \log n}\right)^{2/3}$ we obtain the main result of this paper.

► **Theorem 27.** *The min-cost circulation in a planar multigraph can be found in $O\left((nm)^{2/3} \cdot \frac{\log^{5/3} n \log^{1/3} m}{\log^{4/3} \log n} \cdot \log(nC)\right)$ time.*

References

- 1 Mudabir Kabir Asathulla, Sanjeev Khanna, Nathaniel Lahn, and Sharath Raghvendra. A Faster Algorithm for Minimum-Cost Bipartite Perfect Matching in Planar 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 457–476, 2018. doi:10.1137/1.9781611975031.31.
- 2 Dimitri P. Bertsekas and Paul Tseng. Relaxation Methods for Minimum Cost Ordinary and Generalized Network Flow Problems. *Operations Research*, 36(1):93–114, 1988. doi:10.1287/opre.36.1.93.
- 3 Glencora Borradaile and Philip N. Klein. An $O(n \log n)$ algorithm for maximum st -flow in a directed planar graph. *J. ACM*, 56(2):9:1–9:30, 2009. doi:10.1145/1502793.1502798.
- 4 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. *SIAM J. Comput.*, 46(4):1280–1303, 2017. doi:10.1137/15M1042929.
- 5 Robert G Busacker and Paul J Gowen. A procedure for determining a family of minimum-cost network flow patterns. Technical report, RESEARCH ANALYSIS CORP MCLEAN VA, 1960.
- 6 Sergio Cabello, Erin W. Chambers, and Jeff Erickson. Multiple-Source Shortest Paths in Embedded Graphs. *SIAM J. Comput.*, 42(4):1542–1571, 2013. doi:10.1137/120864271.
- 7 Michael B. Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. Negative-Weight Shortest Paths and Unit Capacity Minimum Cost Flow in $\tilde{O}(m^{10/7} \log W)$ Time (Extended Abstract). In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 752–771, 2017. doi:10.1137/1.9781611974782.48.
- 8 Samuel I. Daitch and Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 451–460, 2008. doi:10.1145/1374376.1374441.
- 9 Robert B. Dial. Algorithm 360: shortest-path forest with topological ordering [H]. *Commun. ACM*, 12(11):632–633, 1969. doi:10.1145/363269.363610.
- 10 Jack Edmonds and Richard M. Karp. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM*, 19(2):248–264, 1972. doi:10.1145/321694.321699.
- 11 Jeff Erickson. Maximum Flows and Parametric Shortest Paths in Planar Graphs. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 794–804, 2010. doi:10.1137/1.9781611973075.65.
- 12 Shimon Even and Robert Endre Tarjan. Network Flow and Testing Graph Connectivity. *SIAM J. Comput.*, 4(4):507–518, 1975. doi:10.1137/0204043.
- 13 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.
- 14 L. R. Ford and D. R. Fulkerson. Constructing Maximal Dynamic Flows from Static Flows. *Operations Research*, 6(3):419–433, 1958.

- 15 Harold N. Gabow and Robert Endre Tarjan. Faster Scaling Algorithms for Network Problems. *SIAM J. Comput.*, 18(5):1013–1036, 1989. doi:10.1137/0218069.
- 16 Pawel Gawrychowski and Adam Karczmarz. Improved Bounds for Shortest Paths in Dense Distance Graphs. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 61:1–61:15, 2018. doi:10.4230/LIPIcs.ICALP.2018.61.
- 17 Andrew V. Goldberg, Sagi Hed, Haim Kaplan, and Robert E. Tarjan. Minimum-Cost Flows in Unit-Capacity Networks. *Theory Comput. Syst.*, 61(4):987–1010, 2017. doi:10.1007/s00224-017-9776-7.
- 18 Andrew V. Goldberg and Robert E. Tarjan. Finding Minimum-Cost Circulations by Successive Approximation. *Math. Oper. Res.*, 15(3):430–466, 1990. doi:10.1287/moor.15.3.430.
- 19 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. doi:10.1137/0202019.
- 20 Masao Iri. A new method of solving transportation-network problems. *Journal of the Operations Research Society of Japan*, 1960.
- 21 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.
- 22 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.
- 23 William S. Jewell. Optimal Flow through Networks with Gains. *Operations Research*, 10(4):476–499, 1962. URL: <http://www.jstor.org/stable/168050>.
- 24 Adam Karczmarz. Decremental Transitive Closure and Shortest Paths for Planar Digraphs and Beyond. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 73–92, 2018. doi:10.1137/1.9781611975031.5.
- 25 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, British Columbia, Canada, January 23-25, 2005*, pages 146–155, 2005. URL: <http://dl.acm.org/citation.cfm?id=1070432.1070454>.
- 26 Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 505–514, 2013. doi:10.1145/2488608.2488672.
- 27 Nathaniel Lahn and Sharath Raghvendra. A Faster Algorithm for Minimum-Cost Bipartite Matching in Minor-Free Graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 569–588, 2019. doi:10.1137/1.9781611975482.36.
- 28 Yin Tat Lee and Aaron Sidford. Path Finding Methods for Linear Programming: Solving Linear Programs in \tilde{O} (vrnk) Iterations and Faster Algorithms for Maximum Flow. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 424–433, 2014. doi:10.1109/FOCS.2014.52.
- 29 Gary L. Miller and Joseph Naor. Flow in Planar Graphs with Multiple Sources and Sinks. *SIAM J. Comput.*, 24(5):1002–1017, 1995. doi:10.1137/S0097539789162997.
- 30 Yahav Nussbaum. *Network flow problems in planar graphs*. PhD thesis, Tel Aviv University, 2014.
- 31 James B. Orlin. A Faster Strongly Polynomial Minimum Cost Flow Algorithm. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 377–387, 1988. doi:10.1145/62212.62249.

- 32 Thomas L Saaty and Robert G Busacker. *Finite graphs and networks: An introduction with applications*. McGraw-Hill Book Company, 1965.
- 33 Piotr Sankowski. NC algorithms for weighted planar perfect matching and related problems. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 97:1–97:16, 2018. doi:10.4230/LIPIcs.ICALP.2018.97.
- 34 Éva Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–256, 1985. doi:10.1007/BF02579369.
- 35 Nobuaki Tomizawa. On some techniques useful for solution of transportation network problems. *Networks*, 1(2):173–194, 1971.

Global Curve Simplification

Mees van de Kerkhof

Utrecht University, The Netherlands
m.a.vandekerkhof@uu.nl

Irina Kostitsyna

TU Eindhoven, The Netherlands
i.kostitsyna@tue.nl

Maarten Löffler

Utrecht University, The Netherlands
m.loffler@uu.nl

Majid Mirzanezhad

Tulane University, New Orleans, USA
mmirzane@tulane.edu

Carola Wenk

Tulane University, New Orleans, USA
cwenk@tulane.edu

Abstract

Due to its many applications, *curve simplification* is a long-studied problem in computational geometry and adjacent disciplines, such as graphics, geographical information science, etc. Given a polygonal curve P with n vertices, the goal is to find another polygonal curve P' with a smaller number of vertices such that P' is sufficiently similar to P . Quality guarantees of a simplification are usually given in a *local* sense, bounding the distance between a shortcut and its corresponding section of the curve. In this work we aim to provide a systematic overview of curve simplification problems under *global* distance measures that bound the distance between P and P' . We consider six different curve distance measures: three variants of the *Hausdorff* distance and three variants of the *Fréchet* distance. And we study different restrictions on the choice of vertices for P' . We provide polynomial-time algorithms for some variants of the global curve simplification problem, and show NP-hardness for other variants. Through this systematic study we observe, for the first time, some surprising patterns, and suggest directions for future research in this important area.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Curve simplification, Fréchet distance, Hausdorff distance

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.67

Related Version A full version of the paper is available at <http://arxiv.org/abs/1809.10269>.

Funding *Mees van de Kerkhof*: Supported by the Netherlands Organisation for Scientific Research (NWO) under project number 628.011.005.

Maarten Löffler: Partially supported by the Netherlands Organisation for Scientific Research (NWO) under project numbers 614.001.504 and 628.011.005.

Majid Mirzanezhad: Supported by the National Science Foundation grant CCF-1637576.

Carola Wenk: Supported by the National Science Foundation grant CCF-1637576.

1 Introduction

Due to its many applications, *curve simplification* (also known as *line simplification*) is a long-studied problem in computational geometry and adjacent disciplines, such as graphics, geographical information science, etc. Given a polygonal curve P with n vertices, the goal is to find another polygonal curve P' with a smaller number of vertices such that P' is



© Mees van de Kerkhof, Irina Kostitsyna, Maarten Löffler, Majid Mirzanezhad, and Carola Wenk; licensed under Creative Commons License CC-BY

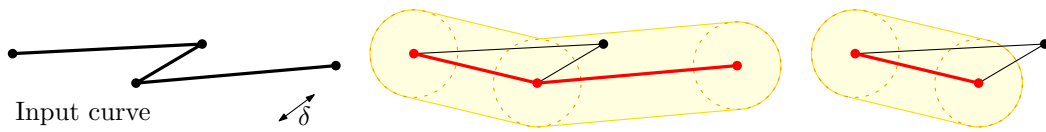
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 67; pp. 67:1–67:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** For a target distance δ , the red curve (middle) is a global simplification of the input curve (left), but it is not a local simplification, since the first shortcut does not closely represent its corresponding curve section (right). The example works for both Hausdorff and Fréchet distance.

sufficiently similar to P . Classical algorithms for this problem famously include a simple recursive scheme by Douglas and Peucker [16], and a more involved dynamic programming approach by Imai and Iri [21]; both are frequently implemented and cited. Since then, numerous further results on curve simplification, often in specific settings or under additional constraints, have been obtained [1, 2, 6, 11, 12, 14, 8, 18, 20].

Despite its popularity, the Douglas-Peucker algorithm comes with no provable quality guarantees. The method by Imai and Iri, though slower, was introduced as an alternative which does supply guarantees: it finds an optimal shortest path in a graph in which potential shortcuts are marked as either *valid* or *invalid*, based on their distance to the corresponding sections of the input curve. However, Agarwal et al. [2] note that the Imai-Iri algorithm does not actually globally optimize any distance measure between the original curve P and the simplification P' . This work initiated a more formal study of curve simplification; van Kreveld et al. [24] systematically show that both Douglas-Peucker and Imai-Iri may indeed produce far-from-optimal results.

This raises a question of what it means for a simplification to be optimal. We may view it as a dual-optimization problem: we wish to minimize the number of vertices of P' given a constraint on its similarity to P . This depends on the distance measure used; popular curve distance measures include the *Hausdorff* and *Fréchet* distances (variants and formal definitions are discussed in Section 2.1). However, the difference in interpretation between Agarwal et al. and Imai and Iri lies not so much in the choice of distance measure, but rather what exactly it is applied to. In fact, the Imai-Iri algorithm is optimal in a *local* sense: it outputs a subsequence of the vertices of P such that the Hausdorff distance between each shortcut and *its corresponding section of the input* is bounded: each shortcut approximates the section of P between the vertices of the shortcut.

In this work, we underline this difference by using the term *global* simplification when a bound on a distance measure must be satisfied between P and P' (formal definition in Section 2.3), and *local* simplification when a bound on a distance measure must be satisfied between each edge of P' and its corresponding section of P . Clearly, a local simplification is also a global simplification, but the reverse is not necessarily true, see Figure 1. Both local and global simplifications have their merits: one can imagine situations where it is important that each segment of a simplified curve is a good representation of the curve section it replaces, but in other applications (e.g., visualization) it is really the similarity of the overall result to the original that matters. Most existing work on curve simplification falls in the *local* category. In this work, we focus on *global* curve simplification.

1.1 Existing Work on Global Curve Simplification

Surprisingly, only a few results on simplification under global distance measures are known [2, 7, 10, 24]; consequently, what makes the problem difficult is not well understood.

Agarwal et al. [2] first consider the idea of global simplification. They introduce what they call a *weak simplification*: a model in which the vertices of the simplification are not restricted to be a subset of the input vertices, but can lie anywhere in the ambient space.¹ Interestingly, they compare this to a *local* simplification where vertices are restricted to be a subset of the input. We may interpret a combination of two of their results (Theorem 1.2 and Theorem 4.1) as an approximation algorithm for global curve simplification with unrestricted vertices under the Fréchet distance: for a given curve P and threshold δ one can compute, in $O(n \log n)$ time, a simplification P' which has at most the number of vertices of an optimal simplification with threshold $\delta/8$.

Bereg et al. [7] first explicitly consider global simplification in the setting where vertices are restricted to be a subsequence of input vertices, but using the *discrete Fréchet distance*: a variant of the Fréchet distance which only measures distances between vertices (refer to Section 2.1). They show how to compute an optimal simplification where vertices are restricted to be a subsequence in $O(n^2)$ time, and they give an $O(n \log n)$ time algorithm for the setting where vertices may be placed freely.

Van Kreveld et al. [24] consider the same (global distance, but vertices should be a subsequence) setting, but for the continuous Fréchet and Hausdorff distances. They give polynomial-time algorithms for the Fréchet distance and directed Hausdorff distance (from simplification curve to input curve), but they show the problem is NP-hard for the directed Hausdorff distance in the opposite direction and for the undirected Hausdorff distance. Recently, Bringmann and Chaudhury [10] improved their result for the Fréchet distance to $O(n^3)$, and also give a conditional cubic lower bound.

Finally, we mention there is earlier work which does not explicitly study simplification under global distance measures, but contains results that may be reinterpreted as such. Guibas et al. [19] provide algorithms for computing minimum-link paths that stab a sequence of regions in order. One of the variants, presented in Theorems 10 and 14 of [19], computes what may be seen as an optimal simplification under the Fréchet distance with no vertex restrictions, i.e., the same setting that was studied by Agarwal et al., in $O(n^2 \log^2 n)$ time.

2 Classification

We aim to provide a systematic overview of curve simplification problems under global distance measures. To this end, we have collected known results and arranged them in a table (Table 1), and provide several new results to complement these (refer to Section 2.4). This allows us for the first time to observe some surprising patterns, and it suggests directions for future research in this important area. We first discuss the dimensions of the table.

2.1 Distance Measures

For our study, we consider six different curve distance measures: three variants of the *Hausdorff* distance and three variants of the *Fréchet* distance. These are among the most popular curve distance measures in the algorithms literature. The Hausdorff distance captures the maximum distance from a point on one curve to a point on the other curve. The variants of the Hausdorff distance we consider are the directed Hausdorff distance from the input to the output, the directed Hausdorff distance from the output to the input, and the undirected

¹ We choose not to adopt the terms *weak* and *strong* in this context because we will also distinguish an intermediate model, and to avoid confusion with the *weak Fréchet* distance; refer to Section 2.2.

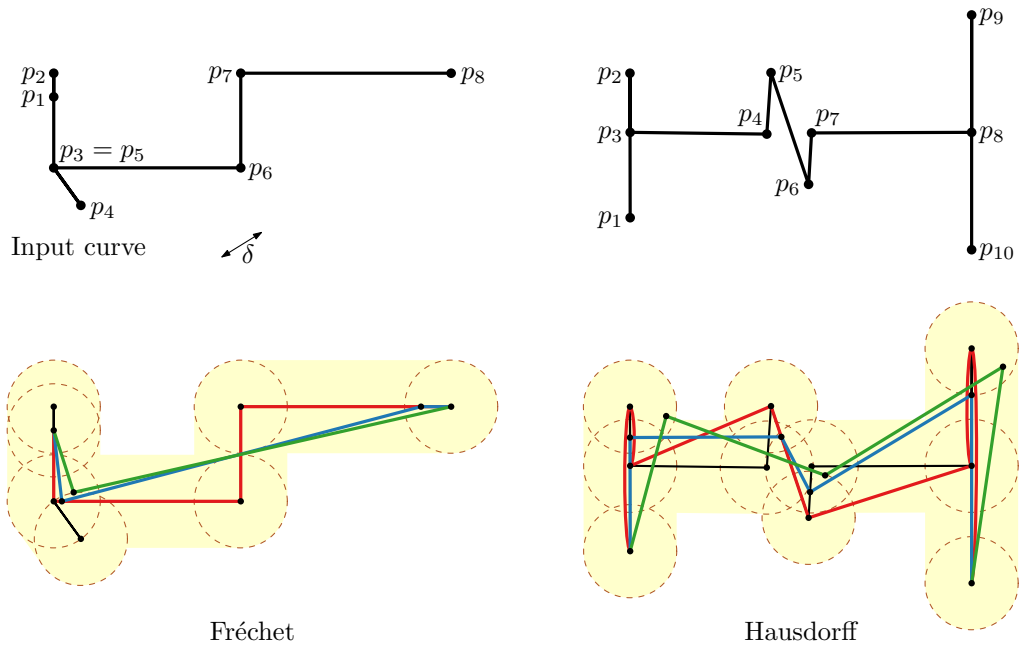


Figure 2 Globally simplified curves under Fréchet distance (left) and Hausdorff distance (right). The vertex-restricted case (in red) requires 5 vertices for Fréchet distance and 8 vertices for the Hausdorff distance. The curve-restricted case (in blue) requires 4 vertices for Fréchet distance and 6 vertices for the Hausdorff distance. The non-restricted case (in green) requires only 3 vertices for Fréchet distance and only 5 vertices for the Hausdorff distance. The δ -neighborhoods for the original curves are shown in yellow.

(or bidirectional) Hausdorff distance. The Fréchet distance captures the maximum distance between a pair of points traveling along the two curves simultaneously. We now formally define all six distance measures.

Let $P = \langle p_1, p_2, \dots, p_n \rangle$ be the input polygonal curve. We treat P as a continuous map $P : [1, n] \rightarrow \mathbb{R}^d$, where $P(i) = p_i$ for integer i , and the i -th edge is linearly parametrized as $P(i + \lambda) = (1 - \lambda)p_i + \lambda p_{i+1}$. We write $P[s, t]$ for the subcurve between $P(s)$ and $P(t)$ and denote the *shortcut*, i.e., the straight line connecting them, by $\langle P(s)P(t) \rangle$.

The *Fréchet distance* between two polygonal curves P and Q , with n and m vertices, respectively, is $F(P, Q) = \inf_{(\sigma, \theta)} \max_t \|P(\sigma(t)) - Q(\theta(t))\|$, where σ and θ are continuous non-decreasing functions from $[0, 1]$ to $[1, n]$ and $[1, m]$, respectively. If σ and θ are continuous but not necessarily monotone, the resulting infimum is called the *weak Fréchet distance*. Finally, the *discrete Fréchet distance* is a variant where σ and θ are discrete functions from $\{1, \dots, k\}$ to $\{1, \dots, n\}$ and $\{1, \dots, m\}$ with the property that $|\sigma(i) - \sigma(i + 1)| \leq 1$.

The *directed Hausdorff distance* between two polygonal curves (or more generally, compact sets) P and Q is defined as $\vec{H}(P, Q) = \max_{p \in P} \min_{q \in Q} \|p - q\|$. The *undirected Hausdorff distance* is then simply the maximum over the two directions: $H(P, Q) = \max\{\vec{H}(P, Q), \vec{H}(Q, P)\}$.

2.2 Vertex Restrictions

Once we have fixed the distance measure and agreed that we wish to apply it globally, one important design decision still remains to be made. Traditional curve simplification algorithms consider the (polygonal) input curve P to be a sequence of points, and produce

■ **Table 1** Known and new results (in blue) for the GCS problem under global distance measures.

Distance	Vertex-restricted (\mathcal{V})	Curve-restricted (\mathcal{C})	Non-restricted (\mathcal{N})
$\overleftarrow{\mathbf{H}}(P, \delta)$	strongly NP-hard [24]	weakly NP-hard (Thm 5)	?
$\overrightarrow{\mathbf{H}}(P, \delta)$	$O(n^4)$ [24] $O(n^3 \log n)$ (Thm 3)	weakly NP-hard (Thm 5)	poly(n) [23]
$\mathbf{H}(P, \delta)$	strongly NP-hard [24]	strongly NP-hard (Cor 14)	strongly NP-hard (Thm 13)
$\mathbf{F}(P, \delta)$	$O(mn^5)$ [24] $O(n^3)$ [22] $O(n^3)$ [10]	$O(n)$ in \mathbb{R}^1 (Thm 7) weakly NP-hard in \mathbb{R}^2 (Thm 5)	$O(n^2 \log^2 n)$ in \mathbb{R}^2 [19] $O(n \log n)$ (1, 8)-approx [2] $O^*(n^2 \log n \log \log n)$ (2, $1 + \varepsilon$)-approx (Thm 11)
$\mathbf{dF}(P, \delta)$	$O(n^2)$ [7]	$O(n^3)$ (Thm 6)	$O(n \log n)$ [7]
$\mathbf{wF}(P, \delta)$	$O(n^3)$ (Thm 2)	weakly NP-hard (Thm 5)	(2, $1 + \varepsilon$)-approx (Cor 12)

as output P' a subsequence of this sequence. However, if we measure the distance globally, there may be no strong reason to restrict the family of acceptable output curves so much: the distance measure already ensures the similarity between input and output curves, so perhaps we may allow a more free choice of vertex placement. Indeed, several results under this more relaxed viewpoint exist, as discussed in Section 1.1. Here, we choose to investigate three increasing levels of freedom: (1) *vertex-restricted* (\mathcal{V}), where vertices of P' have to be a subsequence of vertices of P ; (2) *curve-restricted* (\mathcal{C}), where vertices of P' can lie anywhere on P but have to respect the order along P ; and (3) *non-restricted* (\mathcal{N}), where vertices of P' can be anywhere in the ambient space. Figure 2 illustrates the difference between the three models. The third category does not make sense for local curve simplification, but is very natural for global curve simplification. Observe that when the vertices of a simplified curve have more freedom, the optimal simplified curve never has more, but may have fewer, vertices.

2.3 Global Curve Simplification Overview

We are now ready to formally define a class of global curve simplification problems. When $D(\cdot, \cdot)$ denotes a distance measure between curves (e.g., the *Hausdorff* or *Fréchet* distance), the *global curve simplification* (GCS) problem asks what is the smallest number k such that there exists a curve P' with at most k vertices, chosen either as a subsequence of the vertices of P (variant \mathcal{V}), as a sequence of points on the edges of P in the correct order along P (variant \mathcal{C}), or chosen anywhere in \mathbb{R}^d (variant \mathcal{N}) and such that $D(P, P') \leq \delta$, for a given threshold δ . In all cases, we require that P and P' start at the same point and end at the same point.

Table 1 summarizes results for the different variants of the GCS problem obtained by instantiating D with the Hausdorff or Fréchet distance measures and by applying a vertex restriction R . Here $R \in \{\mathcal{V}, \mathcal{C}, \mathcal{N}\}$, and D is either the undirected Hausdorff distance \mathbf{H} , the directed Hausdorff distance $\overleftarrow{\mathbf{H}}(P, \delta)$ from P to P' , the directed Hausdorff distance $\overrightarrow{\mathbf{H}}(P, \delta)$ from P' to P , the Fréchet distance \mathbf{F} , the discrete Fréchet distance \mathbf{dF} , or the weak Fréchet distance \mathbf{wF} . Throughout the paper we use $D_R(P, \delta)$ to denote a curve P' that is the optimal R -restricted simplification of P with $D(P, P') \leq \delta$.

Since GCS is a dual-optimization problem, we call an algorithm an (α, β) -approximation if it computes a solution with distance at most $\beta\delta$ and uses at most α times more shortcuts than the optimal solution for distance δ .

2.4 New Results

In order to provide a thorough understanding of the different variants of the GCS problem we provide several new results. In some cases these are straightforward adaptations of known results, in other cases they require deeper ideas. Additional lemmas, theorems, and proofs are available in the full version of this paper [22]. We give polynomial time algorithms for finding $wF_{\mathcal{V}}(P, \delta)$, the vertex-restricted GCS under the weak Fréchet distance (Section 3, Theorem 2), and $wF_{\mathcal{V}}(P, \delta)$, the vertex-restricted GCS under the strong Fréchet distance (see [22]). In Section 4 we consider the vertex-restricted problem under the directed Hausdorff distance from P' to P (that is, to find $\vec{H}_{\mathcal{V}}(P, \delta)$), originally considered by van Kreveld et al. [24], and we provide an algorithm with an improved runtime of $O(n^3 \log n)$ (Theorem 3). In Section 5 we prove that solving the curve-restricted GCS is NP-hard for almost all distance measures considered in this paper except for the discrete Fréchet distance (Theorem 6) and strong Fréchet distance in \mathbb{R}^1 (Theorem 7) for which we present polynomial time algorithms. To the best of our knowledge, these are the first results in the curve-restricted setting under global distance measures. Finally, in Section 8, we give a $(2, 1 + \varepsilon)$ -approximation algorithm for $F_{\mathcal{N}}(P, \delta)$, the non-restricted GCS under the Fréchet distance, which runs in $O^*(n^2 \log n \log \log n)$ time, where O^* hides factors polynomial in $1/\varepsilon$ (Theorem 11). The same result also holds for $wF_{\mathcal{N}}(P, \delta)$ (Corollary 12). In Section 9 we show that the non-restricted GCS problem becomes NP-hard when we consider the Hausdorff distance (Theorem 13).

2.5 Discussion

With both the existing work and our new results in place, we now have a good overview of the complexity of the different variants of the GCS problem, see Table 1.

Observe that the curve-restricted variants seem to generally be harder than both the vertex-restricted and the non-restricted variants. That means that, on the one hand, broadening the search space from the vertex-restricted to the curve-restricted case makes the problem harder. But on the other hand it does not give unrestricted freedom of choice, which in turn enables the development of efficient algorithms for the unrestricted case.

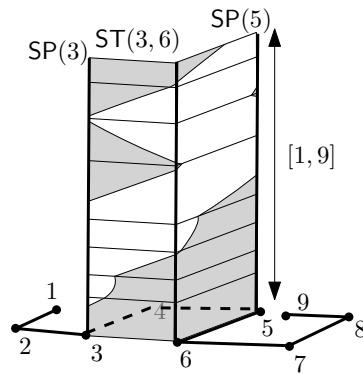
Another interesting pattern can be observed for the Hausdorff distance measures. The direction of the Hausdorff distance makes a significant difference in whether the corresponding GCS problem is NP-hard or polynomially solvable. The GCS problem for the undirected Hausdorff distance is at least as hard as for the directed Hausdorff distance from the input curve to the simplification.

Drawing upon the above observations we make the following conjecture:

► **Conjecture 1.** *The curve-restricted and non-restricted GCS problems for $\overleftarrow{H}(P, \delta)$ are strongly NP-hard.*

3 Freespace-Based Algorithms for Fréchet Simplification

We use the free space diagram between P and its shortcut graph G to solve the vertex-restricted GCS problem under the weak and strong Fréchet distances in $O(n^3)$ time and space. This is related to *map-matching* [4, 9], however in our case we need to compute *shortest* paths in the free space that correspond to *simple* paths in G . While map-matching for closed simple paths is NP-complete [25], we exploit the DAG property of G to develop efficient algorithms. The proof for the strong Fréchet distance can be found in [22].



■ **Figure 3** A free space diagram $FSD_\delta(P, G)$ with strips and spines.

3.1 Shortcut DAG and Free Space Diagram

Let $G = (V, E)$ be the *shortcut DAG* of P , where $V = \{1, \dots, n\}$ and $E = \{(u, v) \mid 1 \leq u < v \leq n\}$. Each $v \in V$ is embedded at p_v and each edge $e = (u, v) \in E$ as a straight line shortcut is linearly parameterized as $e(t) = (1 - t)p_u + tp_v$ for $t \in [0, 1]$. We consider the parameter space of G to be $E \times [0, 1]$.

Now, let $\delta > 0$, and consider the joint parameter space $[1, n] \times E \times [0, 1]$ of P and G . Any $(s, e, t) \in [1, n] \times E \times [0, 1]$ is called *free* if $\|P(s) - e(t)\| \leq \delta$, and the union of all free points is referred to as the *free space*. For brevity, we write $(s, e(t))$ instead of (s, e, t) , and if $e(t) = v \in V$ we write (s, v) . The *free space diagram* $FSD_\delta(P, G)$ consists of all points in $[1, n] \times E \times [0, 1]$ together with an annotation for each point whether it is free or not. In the special case that the graph is a polygonal curve Q with m vertices, then $FSD_\delta(P, Q)$ consists of $(n - 1) \times (m - 1)$ cells in the domain $[1, n] \times [1, m]$. A monotone path from $(1, 1)$ to (n, m) that lies entirely within the free space corresponds to a pair of monotone reparameterizations (σ, θ) that witness $F(P, Q) \leq \delta$. Such a *reachable path* can be computed using dynamic programming in $O(mn)$ time [5]. If one drops the monotonicity requirement for the path, one obtains a witness for $wF(P, Q) \leq \delta$.

The free space diagram $FSD_\delta(P, G)$ consists of one *cell* for each edge in P and each edge in G . The free space in such a cell is convex. The boundary of a cell comprises four line segments that each contain at most one *free space interval*. $FSD_\delta(P, G)$ is composed of spines and strips. For any $v \in V$ and $e \in E$ we call $SP(v) = [1, n] \times v$ a *spine* and $ST(e) = [1, n] \times e \times [0, 1]$ a *strip*. We denote the free space within spines and strips as $SP_\delta(v) = \{(s, v) \mid 1 \leq s \leq n, \|P(s) - p_v\| \leq \delta\}$ and $ST_\delta(e) = \{(s, e(t)) \mid 1 \leq s \leq n, 0 \leq t \leq 1, \|P(s) - e(t)\| \leq \delta\}$. For $(u, v) \in E$, both spines centered at the vertices of the edge are subsets of the strip: $SP(u), SP(v) \subseteq ST(u, v)$, and $SP(u)$ is a subset of all strips with respect to edges incident on u . See Figure 3 for an illustration.

3.2 Weak Fréchet Distance $wF_V(P, \delta)$ in Polynomial Time

Let $P' = wF_V(P, \delta)$ and let $n' = \#P'$ be the number of vertices in P' . Then P' is a path in G , and P' visits an increasing subsequence of vertices in P (or V). From the fact that $wF(P, P') \leq \delta$ we know that there is a path $\mathcal{P} = (\sigma, \theta)$ from $(1, 1)$ to (n, n') in $FSD_\delta(P, P')$ that lies entirely within free space. And since $FSD_\delta(P, P')$ is a subset of $FSD_\delta(P, G)$, the path $\mathcal{P} = (\sigma, \theta)$ is also a path in $FSD_\delta(P, G)$. Here, σ is a reparameterization of P , and θ is a reparameterization of P' , and P' is simple. We call (s, d) in $FSD_\delta(P, G)$ *weakly reachable* if

there exists a path $\mathcal{P} = (\sigma, \theta)$ from $(1, 1)$ to (s, d) in $\text{FSD}_\delta(P, G)$ that lies in free space such that θ is a reparameterization of a simple path from p_1 to some point on an edge in G . We denote the number of vertices on this simple path by $\#\mathcal{P}$, and we call \mathcal{P} *weakly reachable*. We define the cost function $\phi : [1, n] \times V \rightarrow \mathbb{N}$ as $\phi(z, v) = \min_{\mathcal{P}} \#\mathcal{P}$, where the minimum ranges over all weakly reachable paths to (z, v) in the free space diagram. If no such path exists then $\phi(z, v) = \infty$. Note that all points in a free space interval (on the boundary of a free space cell) have the same ϕ -value.

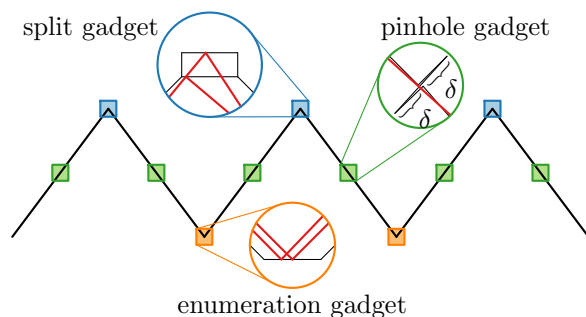
► **Observation 1.** *There is a weakly reachable path \mathcal{P} in $\text{FSD}_\delta(P, G)$ from $(1, 1)$ to (n, n) with $\#\mathcal{P} = \#\text{wF}_V(P, \delta)$ if and only if $\phi(n, n) = \#\text{wF}_V(P, \delta)$.*

Since ϕ is the length of a shortest path, it seems as if one could compute it by simply using a breadth-first propagation. However, one has to be careful because a weakly reachable path \mathcal{P} is only allowed to backtrack *along the path in G that it has already traversed*. We therefore carefully combine two breadth-first propagations to compute the ϕ values for all $I \in \mathcal{I}$, where \mathcal{I} is the set of all (non-empty) free space intervals on all spines $\text{SP}(v)$ for all $v \in V$. For the primary breadth-first propagation, we initialize a queue Q by enqueueing the interval $I \subseteq \text{SP}_\delta(1)$ that contains $(1, 1)$. Once an interval has been enqueueued it is considered *visited*, and it can never become unvisited again. Then we repeatedly extract the next interval I from Q . Assume $I \subseteq \text{SP}_\delta(u)$. For each v from $u + 1$ to n we consider $\text{ST}(u, v)$ and we compute all unvisited intervals $J \subseteq \text{SP}_\delta(u) \cup \text{SP}_\delta(v)$ that are reachable from I with a path in $\text{ST}_\delta(u, v)$. These J can be reached using one more vertex, therefore we set $\phi(J) = \phi(I) + 1$, we insert J into Q , and we store the predecessor $\pi(J) = I$. For each $J \in \text{SP}_\delta(u)$ we then launch a secondary breadth-first traversal to propagate $\phi(J)$ to all unvisited intervals $J' \in \mathcal{I}$ that are reachable from J within the free space of $\text{FSD}_\delta(P, G(\pi(J)))$. Here, $G(\pi(J))$ denotes the projection of the predecessor DAG rooted at $\pi(J)$ onto G , i.e., each interval I in the predecessor DAG is projected to u if $I \subseteq \text{SP}_\delta(u)$. This allows \mathcal{P} to backtrack along the path in G that it has already traversed, *without* increasing ϕ . This secondary breadth-first traversal uses a separate queue Q' , and sets $\phi(J') = \phi(J)$ and $\pi(J') = J$. When this secondary traversal is finished, Q' is prepended to Q , and then the primary breadth-first propagation continues. Once Q is empty, i.e., all intervals have been visited, $\phi(I) = \#\text{wF}_V(P, \delta)$, where $I \subseteq \text{SP}_\delta(n)$ is the interval that contains (n, n) . Backtracking a path \mathcal{P} from n to 1 in the predecessor DAG rooted at $\pi(I)$, and projecting \mathcal{P} onto G , yields the simplified curve P' . This algorithm visits each interval in \mathcal{I} once using nested breadth-first traversals. Since there are $O(n^3)$ free space intervals this takes $O(n^3)$ time and space.

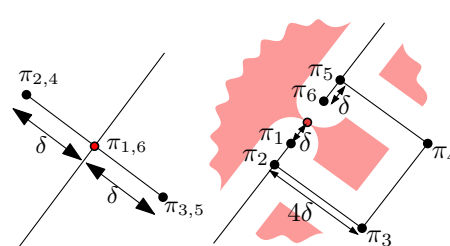
► **Theorem 2.** *Given a polygonal curve P with n vertices and $\delta > 0$, an optimal solution to the vertex-restricted GCS problem under the weak Fréchet distance can be computed in $O(n^3)$ time and space.*

4 Vertex-Restricted GCS under Directed Hausdorff from P' to P

In this section we revisit the GCS problem for $\vec{H}_V(P, \delta)$ considered by Kreveld et al. [24]. We improve on the running time of their $O(n^4)$ time algorithm. First we thicken the input curve P by width δ . This induces a polygon \mathcal{P} with $h = O(n^2)$ holes. Now all we need is to decide whether each shortcut $\langle p_i p_j \rangle$ for all $1 \leq i < j \leq n$ lies entirely within \mathcal{P} or not. To this end, we preprocess \mathcal{P} into a data structure such that for any straight line query ray ρ originating from a point inside \mathcal{P} one can efficiently compute the first point on the boundary of \mathcal{P} hit by ρ . We use the data structure proposed by [13] of size $O(N)$ which can be constructed in time $O(N\sqrt{h} + h^{3/2} \log h + N \log N)$ and which answers queries in



■ **Figure 4** Sketch of the template curve.



■ **Figure 5** Pinhole gadgets π for $F_C(P, \delta)$ (left) and $\overline{H}_C(P, \delta)$ (right). The pinhole is shown in red.

$O(\sqrt{h} \log N)$ time. We have $\Theta(n^2)$ shortcuts $\langle p_i p_j \rangle$ to process and need to examine whether each shortcut lies inside \mathcal{P} or not. If a shortcut lies inside \mathcal{P} then we include it in the edge set of the shortcut graph proposed by Imai and Iri [20]. Otherwise we eliminate the shortcut. We originate a ray at p_i and compute the first point x on the boundary of \mathcal{P} hit by the ray in $O(\sqrt{h} \log N)$ time. If $\|p_i - x\| \geq \|p_i - p_j\|$ then the shortcut lies inside \mathcal{P} , otherwise it does not. Once the edge set of the shortcut graph is constructed, we compute the shortest path in it. As a result we have the following theorem:

► **Theorem 3.** *Given a polygonal curve P with n vertices and $\delta > 0$, an optimal solution to the curve-restricted GCS problem for $\overline{H}_V(P, \delta)$ can be computed in $O(n^3 \log n)$ time using $O(n^2)$ space.*

5 NP-Hardness of Several Curve-Restricted Variants

In this section we construct a template that we use to prove NP-hardness of the curve-restricted GCS problems for most of the distance measures discussed in this paper. The template takes inspiration from the NP-hardness proofs of minimum-link path problems [23]. We believe that this template can be adapted to show hardness of other similar problems.

The template reduces from the SUBSET SUM problem. Given a set of m integers $A = \{a_1, a_2, \dots, a_m\}$ and an integer M , we will construct an instance of the curve-restricted GCS problem such that there exists a subset $B \subset A$ with the total sum of its integers equal to M if and only if there exists a simplified polygonal curve with at most $2m + 1$ vertices.

The input curve P we construct has a zig-zag pattern. It has m *split gadgets* at every other bend of the pattern, $m + 1$ *enumeration gadgets* at the other bends, and $2m$ *pinhole gadgets* halfway through each zig-zag segment (refer to Figure 4).

The construction forces any optimal simplification P' to follow a zig-zag pattern with a vertex on each split and enumeration gadget and no other vertices. The pinhole gadget is named as such because any segment of P' that goes through it is forced to pass through a specific point, called the *pinhole*. This limits the placements of P' 's vertices. The choice of where to place the vertex on each split gadget then corresponds to the choice of including or excluding a given integer in the subset B and the x -coordinate of the vertex on each enumeration gadget encodes the sum of integers in B up to that point. We ensure that the endpoint of P is reachable with at most $2m + 1$ vertices only if B sums to exactly M .

The split and enumeration gadgets always have the same shape, but the shape of the pinhole gadget depends on the distance measure. Pinhole gadgets must be chosen so that the following properties hold:

1. Any segment of P' starting before a pinhole gadget and ending after the pinhole gadget must pass through the pinhole gadget's *pinhole*.
2. It must be impossible to have a segment of P' traverse multiple pinhole gadgets at once.
3. Any segment of P' where the starting vertex u is on a split or enumeration gadget, the segment goes through a pinhole, and the ending vertex v is on the next enumeration or split gadget, must have distance $\leq \delta$ to $P[u, v]$.
4. P must be polynomial in size. Specifically, only a polynomial number of polyline segments can be used and all vertices must have rational coordinates.

In Figure 5 we show pinhole gadgets for Fréchet distance and directed Hausdorff distance directed from P' to P . The gadget for Fréchet distance also works for weak Fréchet distance, undirected Hausdorff distance and directed Hausdorff distance in the other direction. For these latter three distance measures, we note that the pinhole gadget here does not force P' to go through the pinhole but to pass close enough by it instead. This results in there being reachable intervals on the split and enumeration gadgets rather than reachable points. This leads to an expanded version of the first property:

1. The endpoint of any segment of P' starting before a pinhole gadget and ending after the pinhole gadget must have distance less than $\frac{0.5}{2m}$ to the endpoint of the segment with the same starting point that passes exactly through the pinhole and ends on the same segment of P .

If this property holds (as it does for the gadget in Figure 5 (left) under weak Fréchet and Hausdorff distance) the reachable intervals on the gadgets are so small they never overlap, so the reduction still holds. This leads to the following theorems:

► **Theorem 4.** *Given a curve distance measure, if there exists a pinhole gadget that can be inserted in the described template such that the listed properties hold, the curve-restricted GCS problem for that distance measure is NP-hard.*

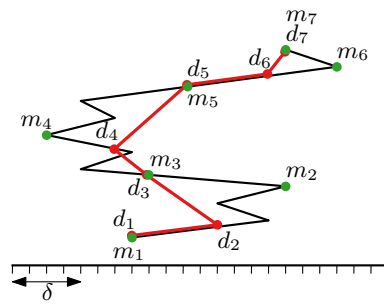
► **Theorem 5.** *The GCS problem for $\overleftarrow{H}_C(P, \delta)$, $\overrightarrow{H}_C(P, \delta)$, $H_C(P, \delta)$, $F_C(P, \delta)$, $wF_C(P, \delta)$ is NP-hard.*

Theorem 4 implies this template may be used to prove curve-restricted simplification under other distance measures NP-hard as well in the future. Since the template reduces from SUBSET SUM it proves the above problems weakly to be NP-hard. For undirected Hausdorff distance, we also prove strong NP-hardness in Corollary 14.

6 Curve-Restricted GCS under Discrete Fréchet Distance

Next we present an $O(n^3)$ -time algorithm for the GCS problem for $dF_C(P, \delta)$. Observe that, given an input curve P , there is only a discrete set of candidate points we need to consider for vertices of the output curve. Let \mathcal{A} be the arrangement of n disks of radius δ centered on the vertices of P , and let $C = \langle c_1, \dots, c_m \rangle$, with $m \in O(n^2)$, be the sequence of intersections between P and \mathcal{A} , in order along P . Observe that under the discrete Fréchet distance, if there exists a curve-restricted simplification $P' = \langle q_1, \dots, q_k \rangle$ of P , then there exists a subsequence of C of length k which is a simplification of P .

Although the approach of Bereg et al. [7] to compute the minimal vertex-restricted simplification of \mathcal{A} does not apply in our case, we can design a dynamic programming algorithm in a similar fashion. Define $K(i, j)$ to be the minimum value k such that there exists a subsequence $\langle c_1, \dots, c_j \rangle$ of length k that has discrete Fréchet distance at most δ to the sequence $\langle p_1, \dots, p_i \rangle$. We will design a dynamic program to calculate all nm values



■ **Figure 6** The time-stamped traversals made by the man m and the dog d . The red lines indicate the dog's jumps.

$K(i, j)$. Specifically, if p_{i-1} and c_j are within distance δ , then

$$K(i, j) = \min \left(K(i-1, j), \min_{1 \leq j' < j} (K(i-1, j') + 1) \right),$$

and $K(i, j) = \infty$ otherwise. This definition immediately gives an $O(n^4)$ algorithm to compute $K(n, m)$. We can improve on this by maintaining a second table with prefix minima $M(i, j) = \min_{1 \leq j' \leq j} K(i, j')$, which can be calculated in constant time per table entry and overall saves a linear factor. The full proof of the following theorem can be found in [22].

► **Theorem 6.** *Given a polygonal curve P with n vertices and $\delta > 0$, an optimal solution to the $\text{dF}_C(P, \delta)$ can be computed in $O(n^3)$ time and $O(n^2)$ space.*

7 GCS Fréchet Distance in One Dimension

In this section we provide a greedy algorithm for the curve-restricted GCS problem in \mathbb{R}^1 under the Fréchet distance. We describe our algorithm using the man-dog terminology that is often used in the literature on Fréchet distance: Initially a man and his dog start at p_1 . The man walks along P until his distance to the dog exceeds δ . Now if there is a turn between the man and the dog, the dog marks its current position and jumps over the turn and stays at distance exactly δ away from the man. If there is no turn in between, the dog just follows the man at distance exactly δ and stops when the man arrives at the next turn or the end. Once they both end the walk at p_n we report the positions marked by the dog as P' . See Figure 6. More details are given in [22].

► **Theorem 7.** *Given a polygonal curve P in \mathbb{R}^1 with n vertices and $\delta > 0$, an optimal solution to the curve-restricted GCS problem under the Fréchet distance can be computed in linear time.*

8 Approximation of Non-Restricted GCS under Fréchet Distance

In this section we present an approximation algorithm for the non-restricted GCS problem that discretizes the feasible space for the vertices of the simplified curve. The idea is to compute a polynomial number of shortcuts in the discretized space, and (approximately) validate for each shortcut whether it is within Fréchet distance δ to a subcurve of P . For every subcurve of P we incrementally add the valid shortcuts to the edge set of a graph G until all the shortcuts have been processed. Once G is built, we compute the shortest path in G and

■ **Algorithm 1** Non-restricted GCS problem for the Fréchet distance.

```

1 forall  $i \in \{1, \dots, n\}$  do Compute  $\mathcal{C}_i(\delta, (\varepsilon\delta/(4\sqrt{d}))$  and  $\mathcal{G}_i$ ;
2  $E \leftarrow \emptyset$ ,  $V \leftarrow \emptyset$ ,  $\mathcal{C}_1 \leftarrow p_1 \cup \mathcal{C}_1$ ,  $\mathcal{C}_n \leftarrow p_n \cup \mathcal{C}_n$ ;
3 forall  $\mathcal{C}_i$  and  $\mathcal{C}_j$ , with  $1 \leq i \leq j \leq n$  do
4   forall  $c_1 \in \mathcal{C}_i$  and  $c_2 \in \mathcal{C}_j$  do
5     if VALIDATE( $\langle c_1 c_2 \rangle, P[i, j]$ ) = true then  $E \leftarrow E \cup \langle c_1 c_2 \rangle$ ,  $V \leftarrow V \cup \{c_1, c_2\}$ ;
6 return the shortest path between  $p_1$  and  $p_n$  in  $G = (V, E)$ .
```

return P' . To speed up the validation for each shortcut, we use a data structure to decide whether the Fréchet distance between a shortcut and a subcurve of P is at most δ . For a better understanding of our algorithm, we introduce some notation. Consider a ball $B(o, r)$ of radius $r > 0$ centered at $o \in \mathbb{R}^d$. Let $\text{Prt}(\mathbb{R}^d, l)$ be a *partitioning* of \mathbb{R}^d into a set of disjoint cells (hypercubes) of side length l that is induced by axis parallel hyperplanes placed consecutively at distance l . For any $1 \leq i \leq n$ we call $\mathcal{C}_i = \mathcal{C}_i(r, l) = \{c \in \text{Prt}(\mathbb{R}^d, l) \mid c \cap B(p_i, r) \neq \emptyset\}$ a *discretization* of $B(p_i, r)$. Let \mathcal{G}_i be the set of *corners* of all cells in \mathcal{C}_i .

As we can see, Algorithm 1 is a straightforward computation of valid shortcuts and shortest path in the graph G . The **VALIDATE** procedure takes a shortcut $\langle c_1 c_2 \rangle$ and a subcurve $P[i, j]$ as arguments and decides (approximately) if $F(\langle c_1 c_2 \rangle, P[i, j]) \leq \delta$. In particular, it returns *true* if $F(\langle c_1 c_2 \rangle, P[i, j]) \leq (1 + \varepsilon/2)\delta$ and *false* if $F(\langle c_1 c_2 \rangle, P[i, j]) > (1 + \varepsilon)\delta$. We implement the **VALIDATE** procedure (line 5) using the data structure in [17]. Let $\#P'$ denote the number of vertices of the polygonal curve P' . The following lemmas imply Theorem 11. More details and proofs are provided in [22].

► **Lemma 8.** *The shortest path P'_{alg} returned by Algorithm 1 exists and $F(P, P'_{\text{alg}}) \leq (1 + \varepsilon)\delta$.*

► **Lemma 9.** *Let $P' = F_{\mathcal{N}}(P, \delta)$ and let P'_{alg} be the curve returned by Algorithm 1. Then $\#P'_{\text{alg}} \leq 2(\#P' - 1)$.*

► **Lemma 10.** *Algorithm 1 runs in $O(\varepsilon^{-d} n \log n (\log^2(1/\varepsilon) \log n + \varepsilon^{-(d+2)} n \log \log n))$ time and uses $O((\varepsilon^{-d} \log^2(1/\varepsilon))n)$ space.*

► **Theorem 11.** *Let P be a polygonal curve with n vertices in \mathbb{R}^d , $\delta > 0$, and $P' = F_{\mathcal{N}}(P, \delta)$. For any $0 < \varepsilon \leq 1$, one can compute in $O^*(n^2 \log n \log \log n)$ time and $O^*(n)$ space a non-restricted simplification P^* of P such that $\#P^* \leq 2(\#P' - 1)$ and $F(P, P^*) \leq (1 + \varepsilon)\delta$. Here, O^* hides factors polynomial in $1/\varepsilon$.*

► **Corollary 12.** *Theorem 11 also holds for the non-restricted GCS under the weak Fréchet distance.*

9 Strong NP-Hardness for Non-Restricted GCS under Undirected Hausdorff Distance

Van Kreveld et al. [24] showed that the vertex-restricted GCS problem is NP-hard for undirected Hausdorff distance by a reduction from Hamiltonian cycle in segment intersection graphs. Their proof can be extended to the curve-restricted and non-restricted case; however, because of the increased freedom in vertex placement we must take care when exact embedding the segment graph: e.g., segments that intersect at arbitrarily small angles could potentially cause coordinates with unbounded bit complexity. For this reason, we here reduce from a

more restricted class of graphs: *orthogonal* segment intersections graphs. Czyzowicz et al. [15] show that Hamiltonian cycle remains NP-complete in 2-connected cubic bipartite planar graphs, and Akiyama et al. [3] prove that every bipartite planar graph has a representation as an intersection graph of orthogonal line segments. Hence, Hamiltonian cycle in orthogonal segment intersection graphs is NP-complete.

We sketch the adapted proof; the full proof can be found in [22]. Let S be a set of n horizontal or vertical line segments in the plane with integer-coordinate endpoints such that $\bigcup S$ forms one connected component. Furthermore, assume that all intersections of segments in S are proper, that is no endpoints of segments in S coincide. Let the input polygonal curve P consist of the subsegments of S , and let P cover all the segments of S (possibly multiple times). That is, the vertices of P are chosen from the set of endpoints and the intersection points of segments in S , and the union of all the links of P equals to the union of the segments in S . Set $\delta = \frac{1}{8}$, and let $D \subset \mathbb{R}^2$ be the Minkowski sum of S and a closed ball of radius δ . A simplification P' with Hausdorff distance at most δ to P must visit the δ -disks around all endpoints of S , while staying inside D . A Hamiltonian path in the intersection graph of S corresponds to a simplification P' with $3n - 1$ vertices. Indeed, since no two δ -disks around the endpoints of the segments in S are visible to each other within D (unless they are endpoints of the same segment), an optimal solution visits the two endpoints of each segment consecutively and has one extra bend to switch to the next segment. This results in three links of P' per segment, except for the first and the last segment to be covered, for which only two links each are needed.

► **Theorem 13.** *The non-restricted GCS problem under undirected Hausdorff distance is strongly NP-hard.*

Since a solution to the reduction never benefits from placing vertices not on P , we also immediately obtain an improvement over Theorem 5 for the case of $H_C(P, \delta)$.

► **Corollary 14.** *The curve-restricted GCS problem under undirected Hausdorff distance is strongly NP-hard.*

References

- 1 M. A. Abam, M. de Berg, P. Hachenberger, and A. Zarei. Streaming Algorithms for Line Simplification. *Discrete & Computational Geometry*, 43(3):497–515, 2010.
- 2 P. K. Agarwal, S. Har-Peled, N. Mustafa, and Y. Wang. Near Linear time approximation algorithm for curve simplification. *Algorithmica*, 42(3–4):203–219, 2005.
- 3 T. Akiyama, T. Nishizeki, and N. Saito. NP-completeness of the hamiltonian cycle problem for bipartite graphs. *Journal of Information Processing*, 3:73–76, 1980.
- 4 H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. *Journal of Algorithms*, 49(2):262–283, 2003.
- 5 H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5(1–2):75–91, 1995.
- 6 G. Barequet, D. Z. Chen, O. Daescu, M. T. Goodrich, and J. Snoeyink. Efficiently Approximating Polygonal Paths in Three and Higher Dimensions. *Algorithmica*, 33(2):150–167, 2002.
- 7 S. Bereg, M. Jiang, W. Wang, B. Yang, and B. Zhu. Simplifying 3D polygonal chains under the discrete Fréchet distance. In *Proc. Latin American Symposium on Theoretical Informatics (LATIN)*, Lecture Notes in Computer Science (LNCS, volume 4957), pages 630–641, 2008.
- 8 M. de Berg, M. van Kreveld, and S. Schirra. Topologically Correct Subdivision Simplification Using the Bandwidth Criterion. *Cartography and Geographic Information Systems*, 25(4):243–257, 1998.

- 9 S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On Map-Matching Vehicle Tracking Data. In *Proc. 31st Conference on Very Large Data Bases (VLDB)*, pages 853–864, 2005.
- 10 K. Bringmann and B.R. Chaudhury. Polyline Simplification has Cubic Complexity, 2018. [arXiv:1810.00621](#).
- 11 L. Buzer. Optimal Simplification of Polygonal Chain for Rendering. In *Proc. 23rd Annual ACM Symposium on Computational Geometry (SoCG)*, pages 168–174, 2007.
- 12 S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *International Journal of Computational Geometry & Applications*, 6(1):59–77, 1996.
- 13 B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.
- 14 D. Z. Chen, O. Daescu, J. Hershberger, P. M. Kogge, N. Mi, and J. Snoeyink. Polygonal path simplification with angle constraints. *Computational Geometry*, 32(3):173–187, 2005.
- 15 J. Czyzowicz, E. Kranakis, and J. Urrutia. A Simple Proof of the Representation of Bipartite Planar Graphs As the Contact Graphs of Orthogonal Straight Line Segments. *Information Processing Letters*, 66(3):125–126, 1998.
- 16 D. H. Douglas and T. K. Peucker. Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature. *Cartographica*, 10(2):112–122, 1973.
- 17 A. Driemel and S. Har-Peled. Jaywalking your dog: computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42:1830–1866, 2013.
- 18 M. Godau. A natural metric for curves – computing the distance for polygonal chains and approximation algorithms. In *Proc. 8th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, Lecture Notes in Computer Science book series (LNCS, volume 480), pages 127–136, 1991.
- 19 L. Guibas, J. Hershberger, J. Mitchell, and J. Snoeyink. Approximating polygons and subdivisions with minimum-link paths. *International Journal of Computational Geometry & Applications*, 3(4):383–415, 1993.
- 20 H. Imai and M. Iri. An optimal algorithm for approximating a piecewise linear function. *Journal of Information Processing*, 9(3):159–162, 1986.
- 21 H. Imai and M. Iri. Polygonal Approximations of a Curve – Formulations and Algorithms. In Godfried T. Toussaint, editor, *Computational Morphology: A Computational Geometric Approach to the Analysis of Form*. North-Holland, Amsterdam, 1988.
- 22 M. van de Kerkhof, I. Kostitsyna, M. Löffler, M. Mirzanezhad, and C. Wenk. Global Curve Simplification, 2019. [arXiv:1809.10269](#).
- 23 I. Kostitsyna, M. Löffler, V. Polishchuk, and F. Staals. On the complexity of Minimum-Link path problems. *Journal of Computational Geometry*, 8(2):80–108, 2017.
- 24 M. van Kreveld, M. Löffler, and L. Wiratma. On optimal polyline simplification using the Hausdorff and Fréchet distance. In *Proc. 34th International Symposium on Computational Geometry (SoCG)*, volume 56, pages 1–14, 2018.
- 25 W. Meulemans. Map Matching with Simplicity Constraints, 2013. [arXiv:1306.2827](#).

Trace Reconstruction: Generalized and Parameterized

Akshay Krishnamurthy

College of Information and Computer Sciences, University of Massachusetts, Amherst, USA
akshay@cs.umass.edu

Arya Mazumdar

College of Information and Computer Sciences, University of Massachusetts, Amherst, USA
arya@cs.umass.edu

Andrew McGregor

College of Information and Computer Sciences, University of Massachusetts, Amherst, USA
mcgregor@cs.umass.edu

Soumyabrata Pal

College of Information and Computer Sciences, University of Massachusetts, Amherst, USA
spal@cs.umass.edu

Abstract

In the beautifully simple-to-state problem of trace reconstruction, the goal is to reconstruct an unknown binary string x given random “traces” of x where each trace is generated by deleting each coordinate of x independently with probability $p < 1$. The problem is well studied both when the unknown string is arbitrary and when it is chosen uniformly at random. For both settings, there is still an exponential gap between upper and lower sample complexity bounds and our understanding of the problem is still surprisingly limited. In this paper, we consider natural parameterizations and generalizations of this problem in an effort to attain a deeper and more comprehensive understanding. Perhaps our most surprising results are:

1. We prove that $\exp(O(n^{1/4}\sqrt{\log n}))$ traces suffice for reconstructing arbitrary matrices. In the matrix version of the problem, each row and column of an unknown $\sqrt{n} \times \sqrt{n}$ matrix is deleted independently with probability p . Our results contrasts with the best known results for sequence reconstruction where the best known upper bound is $\exp(O(n^{1/3}))$.
2. An optimal result for random matrix reconstruction: we show that $\Theta(\log n)$ traces are necessary and sufficient. This is in contrast to the problem for random sequences where there is a super-logarithmic lower bound and the best known upper bound is $\exp(O(\log^{1/3} n))$.
3. We show that $\exp(O(k^{1/3} \log^{2/3} n))$ traces suffice to reconstruct k -sparse strings, providing an improvement over the best known sequence reconstruction results when $k = o(n/\log^2 n)$.
4. We show that $\text{poly}(n)$ traces suffice if x is k -sparse and we additionally have a “separation” promise, specifically that the indices of 1’s in x all differ by $\Omega(k \log n)$.

2012 ACM Subject Classification Mathematics of computing → Probability and statistics

Keywords and phrases deletion channel, trace reconstruction, matrix reconstruction

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.68

Funding *Akshay Krishnamurthy*: Supported in part by NSF Award 1763618.

Arya Mazumdar: Supported in part by NSF Awards 1642658 and 1642550.

Andrew McGregor: Supported in part by NSF Award 1637536.

Soumyabrata Pal: Supported in part by NSF Awards 1642658 and 1642550.



© Akshay Krishnamurthy, Arya Mazumdar, Andrew McGregor, and Soumyabrata Pal;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 68; pp. 68:1–68:25

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the trace reconstruction problem, first proposed by Batu et al. [4], the goal is to reconstruct an unknown string $x \in \{0, 1\}^n$ given a set of random subsequences of x . Each subsequence, or “trace”, is generated by passing x through the *deletion channel* in which each entry of x is deleted independently with probability p . The locations of the deletions are not known; if they were, the channel would be an *erasure channel*. The central question is to find how many traces are required to exactly reconstruct x with high probability.

This intriguing problem has attracted significant attention from a large number of researchers [4, 8, 10, 11, 15, 17, 18, 21, 24, 26–28]. In a recent breakthrough, De et al. [11] and Nazarov and Peres [26] independently showed that $\exp(O((n/q)^{1/3}))$ traces suffice where $q = 1 - p$. This bound is achieved by a *mean-based* algorithm, which means that the only information used is the fraction of traces that have a 1 in each position. While $\exp(O((n/q)^{1/3}))$ is known to be optimal amongst mean-based algorithms, the best algorithm-independent lower bound is the much weaker $\Omega(n^{5/4}/\log n)$ [16].

Many variants of the problem have also been considered including: (1) larger alphabets and (2) an average case analysis where x is drawn uniformly from $\{0, 1\}^n$. Larger alphabets are only easier than the binary case, since we can encode the alphabet in binary, e.g., by mapping a single character to 1 and the rest to 0 and repeating for all characters. In the average case analysis, the state-of-the-art result is that $\exp(O(\log^{1/3}(n)))$ traces suffice¹, whereas $\Omega(\log^{9/4} n / \sqrt{\log \log n})$ traces are necessary [15–17]. Very recently, and concurrent with our work, other variants have been studied including a) where the bits of x are associated with nodes of a tree whose topology determines the distribution of traces generated [10] and b) where x is a codeword from a code with $o(n)$ redundancy [8].

In order to develop a deeper understanding of this intriguing problem, we consider fine-grained parameterization and structured generalizations of trace reconstruction. We prove several new results for these variations that shed new light on the problem. Moreover, in studying these settings, we refine existing tools and introduce new techniques that we believe may be helpful in closing the gaps in the fully general problem.

1.1 Our Results

Parametrizations. We begin by considering parameterizations of the trace reconstruction problem. Given the important role that sparsity plays in other reconstruction problems (see, e.g., Gilbert and Indyk [13]), we first study the recovery of sparse strings. Here we prove the following result.

► **Theorem 1.** *If x has at most k non-zeros, $\exp(O((k/q)^{1/3} \log^{2/3} n))$ traces suffice to recover x exactly, with high probability, where $q = 1 - p = \Omega(k^{-1/2} \log^{1/2} n)$ is the retention probability.*

As some points of comparison, note that there is a trivial $\exp(O(k/q + \log n))$ upper bound, which our result improves on with a polynomially better dependence on k/q in the exponent. The best known results for the general case is $\exp(O((n/q)^{1/3}))$ [11, 26] and our result is a strict improvement when $k = o(n/\log^2 n)$. Note that since we have no restrictions on k in the statement, improving upon $\exp(O((k/q)^{1/3}))$ would imply an improved bound in the general setting.

¹ p is assumed to be constant in that work.

Somewhat surprisingly, our actual result is considerably stronger (See Corollary 7 for a precise statement). We also obtain $\exp(O((k/q)^{1/3} \log^{2/3} n))$ sample complexity in an asymmetric deletion channel, where each 0 is deleted with probability exponentially close to 1, but each 1 is deleted with probability $p = 1 - q$. With such a channel, all but a vanishingly small fraction of the traces contain only 1s, yet we are still able to exactly identify the location of every 0. Since we can accommodate $k = \Theta(n)$ this result also applies to the general case with an asymmetric channel, yielding improvements over De et al. [11] and Nazarov and Peres [26].

We elaborate more on our techniques in the next section, but the result is obtained by establishing a connection between trace reconstruction and learning binomial mixtures. There is a large body of work devoted to learning mixtures [1, 2, 5, 7, 9, 12, 14, 19, 20, 25] where it is common to assume that the mixture components are well-separated. In our context, separation corresponds to a promise that each pair of 1s in the original string is separated by a 0-run of a certain length. Our second result concerns strings with a separation promise.

► **Theorem 2.** *If x has at most k 1s and each 1 is separated by 0-run of length $\Omega(k \log n)$, then, with $p = 1/2$, $\text{poly}(n)$ traces suffice to recover x with high probability.*

Note that reconstruction with $\text{poly}(n)$ traces is straightforward if every 1 is separated by a 0-run of length $\Omega(\sqrt{n \log n})$; the basic idea is that we can identify which 1s in a collection of traces correspond to the same 1 in the original sequence and then we can use the indices of these 1s in their respective traces to infer the index of the 1 in the original string. However, reducing to $\Omega(k \log n)$ separation is rather involved and is perhaps the most technically challenging result in this paper.

Here as well, we actually obtain a slightly stronger result. Instead of parameterizing by the sparsity and the separation, we instead parameterize by the number of runs, and the run lengths, where a run is a contiguous sequence of the same character. We require that each 0-run has length $\Omega(r \log n)$, where r is the total number of runs. Note that this parameterization yields a stronger result since r is at most $2k + 1$ if the string is k sparse, but it can be much smaller, for example if the 1-runs are very long. On the other hand, the best lower bound, which is $\Omega(n^{5/4}/\log n)$ [16], considers strings with $\Omega(n)$ runs and run length $O(1)$.

As our last parametrization, we consider a sparse testing problem. We specifically consider testing whether the true string is x or y , with the promise that the Hamming distance between x and y , $\Delta(x, y)$, is at most $2k$. This question is naturally related to sparse reconstruction, since the difference sequence $x - y \in \{-1, 0, 1\}^n$ is $2k$ sparse, although of course neither string may be sparse on its own. Here we obtain the following result.

► **Theorem 3.** *For any pair $x, y \in \{0, 1\}^n$ with $\Delta(x, y) \leq 2k$, $\exp(O(k \log n))$ traces suffice to distinguish between x and y with high probability.*

Generalizations. Turning to generalizations, we consider a natural multivariate version of the trace reconstruction problem, which we call *matrix reconstruction*. Here we receive matrix traces of an unknown binary matrix $X \in \{0, 1\}^{\sqrt{n} \times \sqrt{n}}$, where each matrix trace is obtained by deleting each row and each column with probability p , independently. Here the deletion channel is much more structured, as there are only $2\sqrt{n}$ random bits, rather than n in the sequence case. Our results show that we can exploit this structure to obtain improved sample complexity guarantees.

In the worst case, we prove the following theorem.

► **Theorem 4.** *For the matrix deletion channel with deletion probability p ,*

$$\exp(O(n^{1/4}\sqrt{p\log n/q}))$$

traces suffice to recover an arbitrary matrix $X \in \{0, 1\}^{\sqrt{n} \times \sqrt{n}}$.

While no existing results are directly comparable, it is possible to obtain $\exp(O(n^{1/3} \log n))$ sample complexity via a combinatorial result due to Kós et al. [22]. This agrees with the results from the sequence case, but is obtained using very different techniques. Additionally, our proof is constructive, and the algorithm is actually mean-based, so the only information it requires are estimates of the probabilities that each received entry is 1. As we mentioned, for the sequence case, both Nazarov and Peres [26] and De et al. [11] prove a $\exp(\Omega(n^{1/3}))$ lower bound for mean-based algorithms. Thus, our result provides a strict separation between matrix and sequence reconstruction, at least from the perspective of mean-based approaches.

Lastly, we consider the random matrix case, where every entry of X is drawn iid from $\text{Ber}(1/2)$. Here we show that $O(\log n)$ traces are sufficient.

► **Theorem 5.** *For any constant deletion probability $p < 1$, $O(\log n)$ traces suffice to reconstruct a random $X \in \{0, 1\}^{\sqrt{n} \times \sqrt{n}}$ with high probability over the randomness in X and the channel.*

This result is optimal, since with $o(\log n)$ traces, there is reasonable probability that a row/column will be deleted from all traces, at which point recovering this row/column is impossible. The result should be contrasted with the analogous results in the sequence case. For sequences, the best results for random strings is $\exp(O(\log^{1/3} n))$ [17] and $\Omega(\log^{9/4} n / \sqrt{\log \log n})$ [16]. In light of the lower bound for sequences, it is surprising that matrix reconstruction admits $O(\log n)$ sample complexity.

1.2 Our Techniques

To prove our results, we introduce several new techniques in addition to refining and extending many existing ideas in prior trace reconstruction results.

Theorem 1 is proved via a reduction from trace reconstruction to learning the parameters of a mixture of binomial distributions. Surprisingly, this natural connection does not seem to have been observed in the earlier literature. We then use a generalization of a complex-analytic approach introduced by De et al. [11] and Nazarov and Peres [26] to prove a bound on the sample complexity of learning a binomial mixture. This generalization is to move beyond the analysis of Littlewood polynomials, i.e., polynomials with $\{-1, 0, 1\}$ coefficients, to the case where coefficients have bounded precision. The generalization is not difficult. This is our simplest result to prove but we consider the final result to be revealing as it shows that sparsity plays a more important role than length in the complexity of trace reconstruction.

Our most technically involved result is Theorem 2. This is proved via an algorithm that constructs a hierarchical clustering of the individual 1s in all received traces according to their corresponding position in the original string. This clustering step requires a careful recursion, where in each step we ensure no false negatives (two 1s from the same origin are always clustered together) but we have many false positives, which we successively reduce. At the bottom of the recursion, we can identify a large fraction 1s from each 1 in the original string. However, as the recursion eliminates many of the 1s, simply averaging the positions of the surviving fraction leads to a biased estimate. To resolve this, we introduce a de-biasing step which eliminates even more 1s, but ensures the survivors are unbiased, so that we can accurately estimate the location of each 1 in the original string. The initial recursion has $L = \log \log n$ levels, which is critical since the debiasing step involves conditioning on the presence of 2^L 1s in a trace, which only happens with probability $2^{-2^L} = 1/n$.

Theorem 3 leverages combinatorial arguments about k -decks (the multiset of subsequences of a string) due to Krasikov and Roditty [23]. The result demonstrates the utility of these combinatorial tools in trace reconstruction. As further evidence for the utility of combinatorial tools, the connection to k -decks was also used by Ban et al. [3] in independent concurrent work on the deletion channel.

For Theorem 4, we return to the complex-analytic approach and extend the Littlewood polynomial argument to multivariate polynomials. Since the unknown matrices are $\sqrt{n} \times \sqrt{n}$, we can use a natural bivariate polynomial of degree $O(\sqrt{n})$, which yields the improvement. However, the result of Borwein and Erdélyi [6] used in previous work on trace reconstruction applies only to univariate polynomials. Our key technical result is a generalization of their result to accommodate bivariate Littlewood polynomials, which we then use to demonstrate separation.

For Theorem 5, using an averaging argument and exploiting randomness in the original matrix, we construct a statistical test to determine if two rows (or columns) from two different traces correspond to the same row (column) in the original string. We show that this test succeeds with overwhelming probability, which lets us align the rows and columns in all traces. Once aligned, we know which rows/columns were deleted from each trace, so we can simply read off the original matrix X .

Notation. Throughout, n is the length of the binary string being reconstructed, n_0 is the number of 0s, k is the number of 1s, i.e., the *sparsity* or *weight*. For matrices n is the total number of entries, and we focus on square $\sqrt{n} \times \sqrt{n}$ matrices. For most of our results, we assume that n, n_0, k are known since, if not, they can easily be estimated using a polynomial number of traces. Let p denote the deletion probability when the 1s and 0s are deleted with the same probability. We also study a channel where the 1s and 0s are deleted with different probabilities; in this case, p_0 is the deletion probability of a 0 and p_1 is the deletion probability of a 1. We refer to the corresponding channel as the (p_0, p_1) -Deletion Channel or the asymmetric deletion channel. It will also be convenient to define $q = 1 - p, q_0 = 1 - p_0$ and $q_1 = 1 - p_1$ as the corresponding retention probabilities. Throughout, m denotes the number of traces.

2 Sparsity and Learning Binomial Mixtures

We begin with the sparse trace reconstruction problem, where we assume that the unknown string x has at most k 1s. Our analysis for this setting is based on a simple reduction from trace reconstruction to learning a mixture of binomial distributions, followed by a new sample complexity guarantee for the latter problem. This approach yields two new results: first, we obtain an $\exp(O((k/q_1)^{1/3} \log^{2/3} n))$ sample complexity bound for sparse trace reconstruction, and second, we show that this guarantee applies even if the deletion probability for 0s is exponentially close to 1.

To establish our results, we introduce a slightly more challenging channel which we refer to as the *Austere Deletion Channel*. The bulk of the proof analyzes this channel, and we obtain results for the (p_0, p_1) channel via a simple reduction.

► **Theorem 6** (Austere Deletion Channel Reconstruction). *In the Austere Deletion Channel, all but exactly one 0 are deleted (the choice of which 0 to retain is made uniformly at random) and each 1 is deleted with probability p_1 . For such a channel,*

$$m = \exp(O((k/q_1)^{1/3} \log^{2/3} n))$$

traces suffice for sparse trace reconstruction where $q_1 = 1 - p_1$, provided $q_1 = \Omega(\sqrt{k^{-1} \log n})$.

68:6 Trace Reconstruction: Generalized and Parameterized

We will prove this result shortly, but we first derive our main result for this section as a simple corollary.

► **Corollary 7** (Deletion Channel Reconstruction). *For the (p_0, p_1) -deletion channel,*

$$m = q_0^{-1} \exp(O((k/q_1)^{1/3} \log^{2/3} n))$$

traces suffice for sparse trace reconstruction where $q_0 = 1 - p_0$ and $q_1 = 1 - p_1 = \Omega(\sqrt{k^{-1} \log n})$.

Proof. This follows from Theorem 6. By focusing on just a single 0, it is clear that the probability that a trace from the (p_0, p_1) -deletion channel contains at least one 0 is at least q_0 . If among the retained 0s we keep one at random and remove the rest, we generate a sample from the austere deletion channel. Thus, with m samples from the (p_0, p_1) deletion channel, we obtain at least $m q_0$ samples from the austere channel and the result follows. Note that Theorem 1 is a special case where $p_0 = p_1 = p$. ◀

Remarks. First, note that the case where q_1 is constant (a typical setting for the problem) and $k = o(\log n)$ is not covered by the corollary. However, in this case a simpler approach applies to argue that $\text{poly}(n)$ traces suffice: with probability $q_1^k \geq 1/\text{poly}(n)$ no 1s are deleted in the generation of the trace and given $\text{poly}(n)$ such traces, we can infer the original position of each 1 based on the average position of each 1 in each trace. Second, note that the weak dependence on q_0 ensures that as long as $q_0 = 1/\exp(O((k/q_1)^{1/3} \log^{2/3} n))$, we still have the $\exp(O((k/q_1)^{1/3} \log^{2/3} n))$ bound. Thus, our result shows that sparse trace reconstruction is possible even when zeros are retained with exponentially small probability.

Reduction to Learning Binomial Mixtures. We prove Theorem 6 via a reduction to learning binomial mixtures. Given a string x of length n , let r_i be the number of ones before the i^{th} zero in x . For example, if $x = 1001100$ then $r_1 = 1, r_2 = 1, r_3 = 3, r_4 = 3$. Note that the multi-set $\{r_1, r_2, \dots\}$ uniquely determines x , that each $r_i \leq k$, and that the multi-set has size n_0 . The reduction from trace reconstruction to learning binomial mixtures is appealingly simple:

1. Given traces t_1, \dots, t_m from the austere channel, let s_i be the number of leading ones in t_i .
2. Observe that each s_i is generated by a uniform² mixture of $\text{Bin}(r_1, q_1), \dots, \text{Bin}(r_{n_0}, q_1)$ where $q_1 = 1 - p_1$. Hence, learning r_1, r_2, \dots, r_{n_0} from s_1, s_2, \dots, s_m allows us to reconstruct x .

To obtain Theorem 6, we establish the following new guarantee for learning binomial mixtures.

► **Theorem 8** (Learning Binomial Mixtures). *Let \mathcal{M} be a mixture of $d = \text{poly}(n)$ binomials:*

Draw sample from $\text{Bin}(a_i, q)$ with probability α_i

where $0 \leq a_1, \dots, a_d \leq a$ are distinct integers, the values α_i have $\text{poly}(n)$ precision, and $q = \Omega(\sqrt{a^{-1} \log n})$. Then $\exp(O((a/q)^{1/3} \log^{2/3} n))$ samples suffice to learn the parameters exactly with high probability.

² Note that since the r_i are not necessarily distinct some of the binomial distributions are the same.

Proof. Let \mathcal{M}' be a mixture where the samples are drawn from $\sum_{i=1}^d \beta_i \text{Bin}(b_i, q)$, where $0 \leq b_1, \dots, b_d \leq a$ are distinct and the probabilities $\beta_i \in \{0, \gamma, 2\gamma, \dots, 1\}$ where $1/\gamma = \text{poly}(n)$. Consider the variational distance $\sum_i |A_i - B_i|$ between \mathcal{M} and \mathcal{M}' where

$$A_i = \Pr[\text{sample from } \mathcal{M} \text{ is } i] = \sum_{j=1}^d \alpha_j \binom{a_j}{i} q^i (1-q)^{a_j-i}$$

$$B_i = \Pr[\text{sample from } \mathcal{M}' \text{ is } i] = \sum_{j=1}^d \beta_j \binom{b_j}{i} q^i (1-q)^{b_j-i}.$$

We will show that the variational distance between \mathcal{M} and \mathcal{M}' is at least

$$\epsilon = \exp(-O((a/q)^{1/3}(\log 1/\gamma)^{2/3})).$$

Since there are at most $((a+1) \cdot (1/\gamma+1))^d$ possible choices for the parameters of \mathcal{M}' , standard union bound arguments show that

$$O(\log(((a+1) \cdot (1/\gamma+1))^d / \epsilon^2)) = \exp(O((a/q)^{1/3}(\log 1/\gamma)^{2/3}))$$

samples are sufficient to distinguish \mathcal{M} from all other mixtures.

To prove the total variation bound, observe that by applying the binomial formula, for any complex number w , we have

$$\begin{aligned} \sum_{i \geq 0} (A_i - B_i) w^i &= \sum_{i \geq 0} w^i \left(\sum_{j \geq 0} \alpha_j \binom{a_j}{i} q^i (1-q)^{a_j-i} - \sum_{j \geq 0} \beta_j \binom{b_j}{i} q^i (1-q)^{b_j-i} \right) \\ &= \sum_{j \geq 0} (\alpha_j z^{a_j} - \beta_j z^{b_j}) \end{aligned}$$

where $z = qw + (1-q)$. Let $G(z) = \sum_{j \geq 0} (\alpha_j z^{a_j} - \beta_j z^{b_j})$ and apply the triangle inequality to obtain:

$$\sum_{i \geq 0} |A_i - B_i| |w^i| \geq |G(z)|.$$

Note that $G(z)$ is a non-zero degree d polynomial with coefficients in the set

$$\{-1, \dots, -2\gamma, -\gamma, 0, \gamma, 2\gamma, \dots, 1\}.$$

We would like to find a z such that $G(z)$ has large modulus but $|w^i|$ is small, since this will yield a total variation lower bound. We proceed along similar lines to Nazarov and Peres [26] and De et al. [11]. It follows from Corollary 3.2 in Borwein and Erdélyi [6] that there exists $z \in \{e^{i\theta} : -\pi/L \leq \theta \leq \pi/L\}$ such that

$$|G(z)| \geq \gamma \exp(-c_1 L \log(1/\gamma))$$

for some constant $c_1 > 0$. For such a value of z , Nazarov and Peres [26] show that

$$|w| \leq \exp(c_2/(qL)^2)$$

for some constant $c_2 > 0$. Therefore,

$$\sum_{i \geq 0} |A_i - B_i| \exp(ic_2/(qL)^2) \geq \sum_{i \geq 0} |A_i - B_i| |w^i| \geq |G(z)| \geq \gamma \exp(-c_1 L \log(1/\gamma))$$

68:8 Trace Reconstruction: Generalized and Parameterized

For $i > \tau = 6qa$, by an application of the Chernoff bound, $A_i, B_i \leq 2^{-i}$, so we obtain

$$\underbrace{\sum_{i>\tau} 2^{-i} \exp(ic_2/(qL)^2)}_{=T_\tau} + \sum_{i=0}^{\tau} |A_i - B_i| \exp(\tau c_2/(qL)^2) \geq \gamma \exp(-c_1 L \log(1/\gamma)) .$$

$$\sum_{i=0}^{\tau} |A_i - B_i| \geq \frac{\exp(-c_1 L \log(1/\gamma))}{\exp(\tau c_2/(qL)^2)} - \frac{T_\tau}{\exp(\tau c_2/(qL)^2)} \geq \frac{\gamma \exp(-c_1 L \log(1/\gamma))}{\exp(\tau c_2/(qL)^2)} - O(2^{-\tau}) \quad (1)$$

where the second equality follows from the assumption that $c_2/(qL^2) \leq (\ln 2)/2$ (which we will ensure when we set L) since,

$$\frac{T_\tau}{\exp(\tau c_2/(qL)^2)} = \frac{O(1) \cdot 2^{-\tau} \exp(\tau c_2/(qL)^2)}{\exp(\tau c_2/(qL)^2)} = O(2^{-\tau}) .$$

Set

$$L = c \sqrt[3]{\tau/(q^2 \log(1/\gamma))} = c \sqrt[3]{6a/(q \log(1/\gamma))}$$

for some sufficiently large constant c . This ensures that the first term of Eqn. 1 is

$$\exp(-O((a/q)^{1/3} \log^{2/3}(1/\gamma))).$$

Note that

$$\frac{c_2}{qL^2} < \frac{c_2}{qc^2(a/(q \log(1/\gamma)))^{2/3}} \leq \frac{c_2}{c^2} \cdot \left(\frac{\log(1/\gamma)}{aq^{1/2}}\right)^{2/3} \leq \frac{c_2}{c^2} \cdot \left(\frac{\log(1/\gamma)}{aq^2}\right)^{2/3}$$

and so by the assumption that $q = \Omega(\sqrt{\log(1/\gamma)/a})$ we may set the constant c large enough such that $c_2/(qL^2) \leq (\ln 2)/2$ as required. The second term of Eqn. 1 is a lower order term given the assumption from the assumption on q and thus we obtain the required lower bound on the total variation distance. ◀

Theorem 6 now follows from Theorem 8, since in the reduction, we have $d = O(n)$ binomials, one per 0 in x , α_i is a multiple of $1/n_0$ and importantly, we have $a = k$. The key is that we have a polynomial with degree $a = k$ rather than a degree n polynomial as in the previous analysis.

Remark. If all α_i are equal, Theorem 8 can be improved to $\text{poly}(n) \cdot \exp(O((a/p)^{1/3}))$ by using a more refined bound from Borwein and Erdélyi [6] in our proof. This follows by observing that if $\alpha_i = \beta_i = 1/d$, then $\sum_{j \geq 0} (\alpha_j z^{aj} - \beta_j z^{sj})$ is a multiple of a Littlewood polynomial and we may use the stronger bound $|G(z)| \geq \exp(-c_1 L)/d$, see Borwein and Erdélyi [6]. We can also show that the exponential dependence on $a^{1/3}$ in Theorem 8 is necessary.

► **Theorem 9 (Binomial Mixtures Lower Bound).** *There exists subsets*

$$\{a_1, \dots, a_k\} \neq \{b_1, \dots, b_k\} \subset \{0, \dots, a\}$$

such that if $\mathcal{M} = \sum_{i=1}^k \text{Bin}(a_i, 1/2)/k$ and $\mathcal{M}' = \sum_{i=1}^k \text{Bin}(b_i, 1/2)/k$, then $\|\mathcal{M} - \mathcal{M}'\|_{TV} = \exp(-\Omega(a^{1/3}))$. Thus, $\exp(\Omega(a^{1/3}))$ samples are required to distinguish \mathcal{M} from \mathcal{M}' .

Proof. Previous work [11, 26] shows the existence of two strings $x, y \in \{0, 1\}^n$ such that $\sum_i |t_i^x - t_i^y| = \exp(-\Omega(n^{1/3}))$ where t_i^z is the expected value of the i th element (indexed at 0) of a string formed applying the $(1/2, 1/2)$ -deletion channel to the string z . We may assume $\sum_{i \in [n]} x_i = \sum_{i \in [n]} y_i \equiv k$ since otherwise

$$\sum_i |t_i^x - t_i^y| \geq \left| \sum_i t_i^x - \sum_i t_i^y \right| = \left| \sum_{i \in [n]} x_i/2 - \sum_{i \in [n]} y_i/2 \right| \geq 1/2$$

which would contradict the assumption $\sum_i |t_i^x - t_i^y| \neq \exp(-\Omega(n^{1/3}))$.

Consider $\mathcal{M} = \sum_{i=1}^k \text{Bin}(a_i, 1/2)/k$ and $\mathcal{M}' = \sum_{i=1}^k \text{Bin}(b_i, 1/2)/k$, where a_i (b_i) is the number of coordinates preceding the i th 1 in x (y). Note that

$$t_i^x = \sum_{r=1}^k \binom{a_r}{i} / 2^{a_r+1} \quad \text{and} \quad t_i^y = \sum_{r=1}^k \binom{b_r}{i} / 2^{b_r+1},$$

and so

$$\begin{aligned} \|\mathcal{M} - \mathcal{M}'\|_{TV} &= \sum_i |\Pr[\mathcal{M} = i] - \Pr[\mathcal{M}' = i]| \\ &= \sum_i \frac{1}{k} \left| \sum_{r=1}^k \binom{a_r}{i} / 2^{a_r} - \sum_{r=1}^k \binom{b_r}{i} / 2^{b_r} \right| \\ &= \frac{2}{k} \sum_i |t_i^x - t_i^y| = \exp(-\Omega(n^{1/3})), \end{aligned}$$

which proves the result. ◀

3 Well-Separated Sequences

We now prove Theorem 2, showing that $\text{poly}(n)$ traces suffice for reconstruction of a k -sparse string when there are $\Omega(k \log n)$ 0s between each consecutive 1. We call such sequences of 0s the *0-runs* of the string. We also refer to the length of the shortest 0-run as the *gap* g of the string x .

► **Theorem** (Restatement of Theorem 2). *Let x be a k -sparse string of length n and gap at least $ck \log(n)$ for a large enough c . Then $\text{poly}(n)$ traces from the $(1/2, 1/2)$ -Deletion Channel suffice to recover x with high probability.*

In Section 3.1, we present the basic ideas and technical challenges in proving the theorem. We also describe the algorithm in detail and explain how to set the parameters. Full details are presented in Section 3.3. In Section 3.2, we strengthen Theorem 2 to show that $\text{poly}(n)$ traces suffice under the weaker assumption that each 0-run has length $\tilde{\Omega}(r)$ where r is the total number of runs (0-runs + 1-runs). Observe that this is a weaker assumption, since $r \leq 2k + 1$ always, but r can be much less than k .

3.1 A Recursive Hierarchical Clustering Algorithm and Its Analysis: Overview

Let $\{p_u\}_{u=1}^k$ denote the positions (index of the coordinate from the left) of the k 1s in the original string x . Let \mathcal{N} denote the multi-set of all positions of all received 1s and call $N = |\mathcal{N}|$. We will construct a graph G on N vertices where every vertex is associated with a

received 1. We decorate each vertex v with a number $z_v \in \mathcal{N}$, which is the position of the associated received 1. Each vertex v also has an *unknown* label $y_v \in \{1, \dots, k\}$ denoting the corresponding 1 in the *original* string.

At a high level, our approach uses the observed values $\{z_v\}_{v \in V}$ to recover the unknown labels $\{y_v\}_{v \in V}$. Once this “alignment” has been performed, the original string can be recovered easily, since the average of $\{z_v \mathbf{1}\{y_v = u\}\}_{v \in V}$ is an unbiased estimator for $p_u/2$.

A starting observation. Our first observation is a simple fact about binomial concentration, which we will use to define the edge set in G : by the Chernoff bound, with high probability, for every vertex v , if $y_v = u$ then we must have $|z_v - p_u/2| \leq c\sqrt{n \log n}$ for some constant c . Defining the edges in G to be $\{(v, w) : |z_v - z_w| \leq 2c\sqrt{n \log n}\}$ then guarantees that all vertices with $y_v = u$ are connected. This immediately yields an algorithm for the much stronger gap condition $g \geq 4c\sqrt{n \log n}$, since with such separation, no two vertices v, w with $y_v \neq y_w$ will have an edge. Therefore, the connected components reveal the labeling so that $\text{poly}(n)$ traces suffice with $g = \Omega(\sqrt{n \log n})$.

Intuitively, we have constructed a clustering of the received 1s that corresponds to the underlying labeling. To tolerate a weaker gap condition, we proceed recursively, in effect constructing a *hierarchical clustering*. However there are many subtleties that must be resolved.

The first recursion. To proceed, let us consider the weaker gap condition of $g \geq \tilde{\Omega}(k^{1/2}n^{1/4})$. In this regime, G still maintains a *consistency* property that for each u all vertices with $y_v = u$ are in the same connected component, but now a connected component may have vertices with different labels, so that each connected component C identifies a contiguous set $U \subset \{1, \dots, k\}$ of the original 1s. Moreover, due to the sparsity assumption, C must have length, defined as $\max_{v \in C} z_v - \min_{v \in C} z_v$, at most $O(k\sqrt{n \log n})$. Therefore if we can correctly identify every trace that contains the left-most and right-most 1 in U , we can recurse and are left to solve a subproblem of length $O(k\sqrt{n \log n})$. Appealing to our starting observation, this can be done with a gap of $g \geq \tilde{\Omega}(k^{1/2}n^{1/4})$.

The challenge for this step is in identifying every trace that contains the left-most and right-most 1 in U , which we call u_L and u_R respectively. This is important for ensuring a “clean” recursion, meaning that the traces used in the subproblem are generated by passing exactly the same substring through the deletion channel. To solve this problem we use a device that we call a *Length Filter*. For every trace, consider the subtrace that starts with the first received 1 in U and ends with the last received 1 in U (this subtrace can be identified using G). If the trace contains u_L, u_R then the length of this subtrace is $2 + \text{Bin}(L - 2, 1/2)$ where L is the distance between u_L, u_R in the original string. On the other hand, if the subtrace does not contain both end points, then the length is $2 + \text{Bin}(L' - 2, 1/2)$ where $L' \leq L - g$. Since we know that $L \leq \tilde{O}(k\sqrt{n})$ and we are operating with gap condition $g = \tilde{\Omega}(k^{1/2}n^{1/4}) = \tilde{\Omega}(\sqrt{L})$, binomial concentration implies that with high probability we can *exactly* identify the subtraces containing u_L and u_R .

Further recursion. The difficulty in applying a second recursive step is that when $g = o(k^{1/2}n^{1/4})$ the length filter cannot isolate the subtraces that contain the leftmost and rightmost 1s for a block U , so we cannot guarantee a clean recursion. However, substrings that pass through the filter are only missing a short prefix/suffix which upper bounds any error in the indices of the received 1s. We ensure consistency at subsequent levels by incorporating this error into a more cautious definition of the edge set (in fact the additional error is the same order as the binomial deviation at the next level, so it has negligible effect). In this

way, we can continue the recursion until we have isolated each 1 from the original string. The $\Omega(k \log n)$ lower bound on run length arises since the gap at level t of the recursion, g_t , is related to the gap at level $t-1$ via $g_t = \sqrt{k \log n} \cdot g_{t-1}$ with $g_1 = \sqrt{n \log n}$, and this recursion asymptotes at $\Omega(k \log n)$.

The last technical challenge is that, while we can isolate each original 1, the error in our length filter introduces some bias into the recursion, so simply averaging the z_v values of the clustered vertices does not accurately estimate the original position. However, since we have isolated each 1 into pure clusters, for any connected component corresponding to a block of 1s, we can identify *all* traces that contain the first and last 1 in the block. Applying this idea recursively from the bottom up allows us to debias the recursion and accurately estimate all positions.

The algorithm in detail: recursive hierarchical clustering. We now describe the recursive process in more detail. Let us define the thresholds:

$$\tau_1 = \tilde{O}(n^{1/2}), \tau_2 = \tilde{O}(k^{1/2} n^{1/4}), \tau_3 = \tilde{O}(k^{3/4} n^{1/8}), \dots, \tau_D = \tilde{O}(k^{1-1/2^D} n^{1/2^D}),$$

which will be used in the length filter and in the definitions of the edge set. Observe that with $D = \log \log n$, we have $\tau_D = \tilde{O}(k)$. Let $\tilde{x}_1, \dots, \tilde{x}_m$ denote the $m = \text{poly}(n)$ traces. We will construct a sequence of graphs G_1, G_2, \dots, G_D on the vertex sets $V_1 \supset V_2, \dots, \supset V_D$, where each vertex v corresponds to a received 1 in some trace $t_v \in [m]$ and is decorated with its position z_v and the unknown label y_v . The d^{th} round of the algorithm is specified as follows with $z_v^{(1)} = z_v$ and V_1 as the set of all received 1s.

1. Define G_d with edge set $E_d = \{(v, w) : v, w \in V_d \text{ and } |z_v^{(d)} - z_w^{(d)}| \leq \tau_d\}$.
2. Extract $k_d \leq k$ connected components $C_1^{(d)}, \dots, C_{k_d}^{(d)}$ from G_d .
3. For each connected component $C_i^{(d)}$, extract substraces $\{\tilde{x}_j^{(d,i)}\}_{j=1}^m$ where $\tilde{x}_j^{(d,i)}$ is the substring of \tilde{x}_j starting with the first 1 in C_i and ending with the last 1 in C_i . Formally, with $\ell = \min\{z_v : v \in C_i, t_v = j\}$ and $r = \max\{z_v : v \in C_i, t_v = j\}$, we define $\tilde{x}_j^{(d,i)} = \tilde{x}_j[\ell, \dots, r]$.
4. Length Filter: Define $L^{(d,i)} = \max_j \text{len}(\tilde{x}_j^{(d,i)})$. If

$$\text{len}(\tilde{x}_j^{(d,i)}) \leq L^{(d,i)} - \Omega(\sqrt{L^{(d,i)} \log(L^{(d,i)})}),$$

delete all vertices $v \in C_i$ with $t_v = j$. Let V_{d+1} be all surviving vertices.

5. For $v \in V_{d+1}$, define $z_v^{(d+1)} = z_v - \min_{v' \in C_i, t_v = t_{v'}} z_{v'}$.

We analyze the procedure via sequence of lemmas. The first one establishes a basic consistency property: that two 1s originating from the same source 1 are always clustered together.

► **Lemma 10 (Consistency).** *At level d let $V_{d,u} = \{v \in V_d, y_v = u\}$ for each $u \in [k]$. Then with high probability, for each d and u there exists some component $C_i^{(d)}$ at level d such that $V_{d,u} \subset C_i^{(d)}$.*

The next lemma provides a length upper bound on any component, which is important for the recursion. At a high level since we are using a threshold τ_d at level d and the string is k -sparse, no connected component can span more than $k\tau_d$ positions.

► **Lemma 11 (Length Bound).** *For every component $C_i^{(d)}$ at level d , we have $L^{(d,i)} \leq 2k\tau_d$. Moreover if U is a contiguous subsequence of $\{1, \dots, k\}$ with $\bigcup_{u \in U} V_{d,u} \subset C_i^{(d)}$, then $\min_{u \in U} p_u - \max_{u \in U} p_u \leq 2k\tau_d$.*

Finally we characterize the length filter.

► **Lemma 12 (Length Filter).** *For a component $C_i^{(d)}$ at level d , let U be the maximal contiguous subsequence of $\{1, \dots, k\}$ such that $\bigcup_{u \in U} V_{d,u} \subset C_i^{(d)}$. Define $u_L = \arg \min_{u \in U} p_u$ and $u_R = \arg \max_{u \in U} p_u$. Then for any $v \in C_i^{(d)}$, if u_L and u_R are present in t_v , then v survives to round $d + 1$, that is $v \in V_{d+1}$. Moreover, for any $v \in V_{d+1}$, let $p_{\min}(v, U)$ denote the original position of the first 1 from U that is also in the trace t_v . Then we have $p_{\min}(v, u) - p_{u_L} \leq \tilde{O}(\sqrt{k\tau_d})$.*

The lemmas are all interconnected and proved formally in Section 3.3. It is important that the error incurred by the length filter is $\sqrt{k\tau_d} = \tau_{d+1}$ which is exactly the binomial deviation at level $d + 1$. Thus the threshold used to construct G_{d+1} accounts for both the length filter error and the binomial deviation. This property, established in Lemma 12, is critical in the proof of Lemma 10.

For the hierarchical clustering, observe that after $D = \log \log n$ iterations, we have $\tau_D = \tilde{O}(k)$. With gap condition $g = \tilde{\Omega}(k)$ and applying Lemma 10, this means that the connected components at level D each correspond to exactly one 1 in the original string. Moreover since the length filter preserves every trace containing the left-most and right-most 1 in the component, the probability that a subtrace passes through the length filter is at least $1/4$. Hence, after $\log \log n$ levels, the expected number of surviving traces in each cluster is $m/4^{\log \log n} = m/(\log^2 n)$. Thus for each original 1 $u \in \{1, \dots, k\}$, our recursion identifies at least $m/(\log^2 n)$ vertices $v \in V_1$ such that $t_v = u$.

Removing Bias. The last step in the algorithm is to overcome bias introduced by the length filter. The de-biasing process works upward from the bottom of the recursion. Since we have isolated the vertices corresponding to each 1 in the original string, for a component $C_i^{(D-1)}$ at level $D - 1$, we can identify all subtraces that survived to this level that contain the first and last 1 of the corresponding block $U_i^{(D-1)} \subset [k]$. Thus, we can eliminate all subtraces that erroneously passed this length filter.

Working upwards, consider a component $C_i^{(d)}$ that corresponds to a block $U_i^{(d)} \subset [k]$ of 1s in the original string. Since we have performed further clustering, we have effectively partitioned $U_i^{(d)}$ into sub-blocks $U_1^{(d+1)}, \dots, U_s^{(d+1)}$. We would like to identify exactly the subtraces that survived to level d that contain the first and last 1 of $U_i^{(d)}$, but unfortunately this is not possible due to a weak gap condition. However, by induction, we *can exactly* identify all subtraces that survive to level d that contain the first and last 1 of the first and last sub-block of $U_i^{(d)}$, namely $U_1^{(d+1)}$ and $U_s^{(d+1)}$. Thus we can de-bias the length filter at level d by filtering based on a more stringent event, namely the presence of 2^{D-d} nodes. In total to de-bias all length filters above a particular component, we require the presence of $\sum_{d=1}^D 2^{D-d} = O(2^D) = O(\log n)$ nodes, which happens with probability $\Omega(1/n)$. Thus we can debias with only a polynomial overhead in sample complexity. See Figure 1 for an illustration.

3.2 Strengthening to a Parameterization by Runs

We next parameterized the problem by the number of runs, $r = 1 + |\{i \in [n-1] : x_i \neq x_{i+1}\}|$, in the string x being reconstructed. We will argue that if every 0-run has length $\tilde{\Omega}(r)$ then $\text{poly}(n)$ traces suffice. The proof is via a reduction to the k -sparse case in the previous sections.

Let $x' \in \{0, 1\}^{<n}$ be the string formed by replacing every run of 1s in x by a single 1. We first argue that we can reconstruct x' with high probability using $\text{poly}(n)$ traces generated by applying the $(1/2, 1/2)$ -Deletion Channel to x . We will prove this result for

the case $r = \Omega(\log n)$ since otherwise $\text{poly}(n)$ traces is sufficient even with no gap promise.³ Observe that with $m = \text{poly}(n)$ traces, if every 0-run in x has length at least $c \log n$ for some sufficiently large constant $c > 0$, then a bit in every 0-run of x appears in every trace with high probability. Conditioned on this event, no two 1's that originally appeared in different runs of x are adjacent in any trace. Next replace each run of 1s in each trace with a single 1. The end result is that we generate traces that are generated as if we had deleted each 0 in x' with probability $1/2$ and each 1 in x' with probability $1 - 1/2^t \geq 1/2$ where t is the length of the run that the 1 belonged to in x . This channel is not equivalent to the $(1/2, 1/2)$ -Deletion channel, but our analysis for the sparse case continues to hold even if the deletion probability of each 1 is different. Thus we can apply Theorem 2 to recover x' , and the sparsity of x' is at most r . Since the algorithm identifies corresponding 1s in x' in the different traces, we can then estimate the length of the 1-runs in x that were collapsed to each single 1 of x' by looking at the lengths of the corresponding 1-runs in the traces of x before they were collapsed.

► **Theorem 13.** *For the $(1/2, 1/2)$ -Deletion Channel, $\text{poly}(n)$ traces suffice if the lengths of the 0-runs are $\tilde{\Omega}(r)$ where r is the number of runs in x .*

3.3 Sparsity with Gap: Technical Details

This section contains missing details from Section 3. Recall that we have a string $x \in \{0, 1\}^n$ that is k -sparse. We further assume that each 1 in x is separated by a run of g 0s, and we refer to g as the *gap*. Recall that we define $\{p_u\}_{u=1}^k$ as the position of the k 1s in original string, where $p_1 < p_2 < \dots, p_k$. As further notation we refer to the collection of $m = \text{poly}(n)$ traces as $\mathcal{T} = \{\tilde{x}_j\}_{j=1}^m$.

The first level

As a warm up, we show an algorithm called `FindPositions`, that uses $\text{poly}(n)$ traces to reconstruct x exactly with high probability when the gap $g = \Omega(\sqrt{n \log n})$. The algorithm returns the values $\{p_u\}_{u=1}^k$ and crucially uses a *binomial mean estimator*. Given s samples X_1, X_2, \dots, X_s from a binomial distribution $\text{Bin}(n, \frac{1}{2})$ this estimator returns an estimate of n , $\hat{n} = \text{round}\left(\frac{2}{s} \sum_{i=1}^s X_i\right)$, where the round function simply rounds the argument to the nearest integer. From the Hoeffding bound, it is clear that

$$\begin{aligned} \Pr(\hat{n} \neq n) &= \Pr(|\hat{n} - n| \geq 0.5n) = \Pr\left(\left|\frac{1}{s} \sum_{i=1}^s X_i - \frac{n}{2}\right| \geq \frac{1}{4}\right) \\ &\leq 2 \exp\left(-\frac{s}{8n^2}\right) \leq 2 \exp(-n^\epsilon), \end{aligned}$$

as long as $s = 8n^{2+\epsilon}$ for any $\epsilon > 0$.

The algorithm `FindPositions` is displayed in Algorithm 1. Our first result of this section guarantees that with $g = \Omega(\sqrt{n \log n})$ Algorithm 1 recovers x exactly with $\text{poly}(n)$ traces.

► **Theorem 14.** *Algorithm 1 (`FindPositions`) successfully returns the string x from m traces with probability at least $1 - 3n^{-2}$ as long as $m \geq \Omega(n^2 \log n)$ and the gap $g \geq 4\sqrt{2n \log(nm^3)} = \Theta(\sqrt{n \log n})$.*

³ Specifically, if $r = O(\log n)$, with probability at least $1/2^r = 1/\text{poly}(n)$ a trace also has r runs. Given $\text{poly}(n)$ traces with r runs we can estimate each run length because we know the i^{th} run in each such trace corresponds to the i^{th} run in the original string.

68:14 Trace Reconstruction: Generalized and Parameterized

■ Algorithm 1 FindPositions.

Initialize: length of x : n , m traces \mathcal{T} , gap $g > 4\sqrt{2n \log(mn^3)}$.

For each received 1, create a vertex v decorated with tuple (z_v, t_v) where $z_v \in [n]$ is the position of the received 1 and $t_v \in [m]$ is the index of the trace.

Create graph $G = (V, E)$ using vertex set above, and with edges:

$$E = \left\{ (v, w) : |z_v - z_w| \leq \sqrt{2n \log(mn^3)} \right\}$$

Find connected components $C_1, \dots, C_{k'}$ in G (If $k' \neq k$ report failure).

For each connected component C_i , use the binomial mean estimator on $\{z_v\}_{v \in C_i}$ to estimate \hat{p}_i .

Return $\{\hat{p}_i\}_{i=1}^{k'}$.

Proof. First, let us associate with each vertex v an unknown label $y_v \in [k]$ describing the correspondence between this received 1 and a 1 in the original string. The first observation is that if $y_v = u$ then $z_v \sim \text{Bin}(p_u, \frac{1}{2})$ and we always have $p_u \leq n$. Thus, by Hoeffding's inequality and a union bound, we have

$$\Pr[\exists v \in V : |z_v - p_u/2| > \tau] \leq |V| \exp(-2\tau^2/n) \leq \exp(\log(mk) - 2\tau^2/n)$$

And so with $\tau = \sqrt{n \log(mkn^2)}/2$, with probability $1 - n^{-2}$ all z_v values concentrate appropriately.

This event immediately implies that G is *consistent* in the sense that if $y_v = y_{v'}$ then $(v, v') \in E$. Further the gap condition implies the converse property, which we call *purity*: if $y_v \neq y_{v'}$ then $(v, v') \notin E$. Formally, if $y_v \neq y_{v'}$ then

$$\begin{aligned} g/2 &\leq |p_{y_v}/2 - p_{y_{v'}}/2| \\ &\leq |z_v - p_{y_v}/2| + |z_{v'} - p_{y_{v'}}/2| \\ &\leq \sqrt{2n \log(mkn^2)} + |z_v - z_{v'}| \end{aligned}$$

which implies that $|z_v - z_{v'}| \geq g/2 - \sqrt{2n \log(mkn^2)} > \sqrt{2n \log(mn^3)}$. Hence $(v, v') \notin E$.

The above two properties reveal that each connected component can be identified with a single 1 $u \in [k]$ and the component contains exactly the received 1s corresponding to that original one (formally $C_u = \{v : y_v = u\}$). From here we simply use the binomial estimator on each component. First observe that, by a Chernoff bound, with probability $1 - k \exp(-m/36)$, each 1 from the original string appears in at least a 1/3-fraction of the traces, so that $|C_u| \geq m/3$. Then apply the guarantee for the binomial mean estimator along with another union bound over the k positions. Overall the failure probability is

$$n^{-2} + k \exp(-m/36) + 2k \exp\left(\frac{-m}{24n^2}\right)$$

which is at most $3n^{-2}$ with $m \geq 24n^2 \log(2kn^2)$. With this choice, we can tolerate $g = O(\sqrt{n \log n})$. ◀

The recursion

We now use the algorithm **FindPositions** in a recursive manner to estimate the parameters p_1, \dots, p_k even when the gap g is much less than $\sqrt{n \log n}$. Define a series of threshold parameters, to be used in the d^{th} level of recursion:

Algorithm 2 Algorithm RecurGap.

Initialize: Traces $\mathcal{T} = \{\tilde{x}_j\}_{j=1}^m$, gap lower bound $g \geq ck \log^2(n)$.

For each received 1, create vertex v decorated with (z_v, t_v) where $z_v \in [n]$ is the position of the received 1 and $t_v \in [m]$ is the index of the trace.

Set $z_v^{(1)} = z_v, V_1 = V$

for $d = 1, \dots, D$: **do**

 Create graph G_d with vertices V_d and with edges

$$E_1 = \left\{ (v, w) \in V_d : |z_v^{(d)} - z_w^{(d)}| \leq \tau_d/4 \right\}$$

 Extract connected components $C_1^{(d)}, \dots, C_{k_d}^{(d)}$ of G_d .

 For each connected component $C_i^{(d)}$, extract subtraces $\{\tilde{x}_j^{(d,i)}\}_{j=1}^m$ where $\tilde{x}_j^{(d,i)} = \tilde{x}_j[\ell, r]$ and $\ell = \min\{z_v : v \in C_i^{(d)}, t_v = j\}$ and $r = \max\{z_v : v \in C_i^{(d)}, t_v = j\}$.

 Define $L^{(d,i)} = \max_j \text{len}(\tilde{x}_j^{(d,i)})$. If

$$\text{len}(\tilde{x}_j^{(d,i)}) \leq L^{(d,i)} - \sqrt{2L^{(d,i)} \log(kmn)},$$

 delete all vertices $v \in C_i^{(d)}$ with $t_v = j$. Let V_{d+1} be all surviving vertices.

 For $v \in C_i^{(d)} \cap V_{d+1}$, define $z_v^{(d+1)} = z_v - \min\{z_{v'} : v' \in C_i^{(d)}, t_{v'} = t_v\}$.

end for

$$\tau_1 = 4\sqrt{2n \log(mnk)};$$

$$\tau_d = 80\sqrt{k\tau_{d-1} \log(mnk)}, \quad d = 2, \dots, D$$

where the total number of levels is D . Note that, $\tau_d \leq 80^2 \cdot 4\sqrt{2} \cdot k^{1-\frac{1}{2^{d-1}}} n^{\frac{1}{2^d}} \log^{1-1/2^d}(nmk)$. In particular, if $D = O(\log \log n)$ then we have $\tau_D = O(k \log(n))$.

Recall that V is the vertex set for the graph used above, where each vertex corresponds to a received 1 and is associated with an unknown original one y_v . Our main theorem of this section is the following.

► **Theorem 15.** *Assume $g \geq 2\tau_D$ for some $D \leq \log \log(n)$. Then except with probability $1 - 1/n$, Algorithm 2 (RecurGap) with D levels of recursion returns sets $S_1, \dots, S_k \subset V$ such that*

1. *For all $u \in [k]$, $S_u \subset \{v \in V : y_v = u\}$.*
2. *$|S_u| \geq m/\log^5(n)$.*

The theorem follows from the three lemmas stated earlier. Here we re-state the lemmas in detail and provide the proofs.

► **Lemma 16 (Consistency).** *At level d let $V_{d,u} = \{v \in V_d, y_v = u\}$ for each $u \in [k]$. Then with probability $1 - 1/n^2$, for each d and u there exists some component $C_i^{(d)}$ at level d such that $V_{d,u} \subset C_i^{(d)}$.*

► **Lemma 17 (Length Bound).** *At level d , the following holds with probability at least $1 - 1/n^2$: For every component $C_i^{(d)}$ at level d , we have $L^{(d,i)} \leq 2k\tau_d$. Moreover if U is a contiguous subsequence of $\{1, \dots, k\}$ with $\bigcup_{u \in U} V_{d,u} \subset C_i^{(d)}$, then $\min_{u \in U} p_u - \max_{u \in U} p_u \leq 4k\tau_d$.*

► **Lemma 18** (Length Filter). *Assume $m \geq n$. At level d , the following holds with probability at least $1 - 1/n^2$: For a component $C_i^{(d)}$ at level d , let U be the maximal contiguous subsequence of $\{1, \dots, k\}$ such that $\bigcup_{u \in U} V_{d,u} \subset C_i^{(d)}$. Define $u_L = \arg \min_{u \in U} p_u$ and $u_R = \arg \max_{u \in U} p_u$. Then for any $v \in C_i^{(d)}$, if u_L and u_R are present in t_v , then v survives to round $d + 1$, that is $v \in V_{d+1}$. Moreover, for any $v \in V_{d+1}$, let $p_{\min}(v, U)$ denote the original position of the first 1 from U that is also in the trace t_v . Then we have $p_{\min}(v, u) - p_{u_L} \leq 8\sqrt{k\tau_d \log(nmk)}$.*

The proofs of the lemmas are all-intertwined. In the induction step we will assume that all lemmas hold at the previous level of the recursion. Throughout we repeatedly take union bound over all m traces and all up-to- k components, and set the failure probability for each event to be $1/n^2$. In applications of Hoeffding's inequality, this produces a $2 \log(nmk)$ term inside the square root.

Proof of Lemma 17. We proceed by induction. For the base case, by Hoeffding's inequality, we know that for all $v \in V_1$ we have

$$|z_v - p_{y_v}/2| \leq \sqrt{n \log(mkn)} = \tau_1/8$$

except with probability n^{-2} . This means that the position corresponding to a single original 1 u can span at most $\tau_1/4$ positions. If two such regions are merged into a single connected component, then the distance between the regions is at most τ_1 , by construction. Since there are most k such regions, the total length is at most $(k-1)\tau_1 + k\tau_1/4 \leq 2k\tau_1$. The second claim follows from the concentration statement.

For the induction step, assume that the connected components at level $d-1$ have length at most $2k\tau_{d-1}$. Fix a connected component $C_i^{(d-1)}$ and let $u_{i,1}^{(d-1)}$ denote the left-most original 1 present in $C_i^{(d-1)}$ ($u_{i,1}^{(d-1)} = \min\{y_v : v \in C_i^{(d-1)}\}$). By another application of Hoeffding's inequality and using the error guarantee in Lemma 18, we have that

$$\begin{aligned} & |z_v^{(d-1)} - (p_{y_v} - p_{u_{i,1}^{(d-1)}})/2| \\ & \leq |z_v^{(d-1)} - (p_{y_v} - p_{\min}(v, U_i^{(d-1)}))/2| + |p_{\min}(v, U_i^{(d-1)}) - p_{u_{i,1}^{(d-1)}}|/2 \\ & \leq \sqrt{2k\tau_{d-1} \log(mkn)} + 8\sqrt{k\tau_{d-1} \log(mkn)} \leq \tau_d/8 \end{aligned}$$

except with probability n^{-2} . From here, the same argument as in the base case yields the claim. ◀

Proof of Lemma 18. We have two conditions to verify. Fix a component $C_i^{(d)}$ at level d with maximal contiguous subsequence $U \subset [k]$ and recall the definitions $u_L = \arg \min_{u \in U} p_u$ and $u_R = \arg \max_{u \in U} p_u$. By another concentration bound, we know that

$$\forall j : \text{len}(\tilde{x}_j^{(d,i)}) \leq (p_{u_R} - p_{u_L})/2 + \sqrt{(p_{u_R} - p_{u_L}) \log(mnk)}$$

with probability $1 - n^{-2}$. This reveals that:

$$L^{(d,i)} \leq (p_{u_R} - p_{u_L})/2 + \sqrt{(p_{u_R} - p_{u_L}) \log(mnk)}$$

Moreover, for any trace j that contains u_R, u_L the tail bound is two-sided:

$$\forall j \text{ s.t. } u_L, u_R \in \tilde{x}_j^{(d,i)} : \left| \text{len}(\tilde{x}_j^{(d,i)}) - (p_{u_R} - p_{u_L})/2 \right| \leq \sqrt{(p_{u_R} - p_{u_L}) \log(mnk)}.$$

Note that we also have $L^{(d,i)} \geq (p_{u_R} - p_{u_L})/2$ with overwhelming probability as:

$$\begin{aligned} & \Pr[\forall j : \text{len}(\tilde{x}_j^{(d,i)}) \leq (p_{u_R} - p_{u_L})/2] \\ & \leq \prod_{j=1}^m \Pr[\text{len}(\tilde{x}_j^{(d,i)}) \leq (p_{u_R} - p_{u_L})/2 \mid u_R, u_L] \cdot \Pr[u_R, u_L] \\ & \leq \left(\frac{1}{2} \cdot \frac{1}{4}\right)^m = 2^{-3m} \end{aligned}$$

Here we are using the symmetry of the binomial distribution. Thus, with $m \geq n$, the failure probability here is $\exp(-\Omega(n))$, which is negligible.

Using the upper bound on $L^{(d,i)}$ reveals that $\tilde{x}_j^{(d,i)}$ survives, since

$$\begin{aligned} \text{len}(\tilde{x}_j^{(d,i)}) & \geq (p_{u_R} - p_{u_L})/2 - \sqrt{(p_{u_R} - p_{u_L}) \log(mnk)} \\ & \geq L^{(d,i)} - 2\sqrt{(p_{u_R} - p_{u_L}) \log(mnk)} \geq L^{(d,i)} - 2\sqrt{2L^{(d,i)} \log(mnk)}. \end{aligned}$$

For the second condition, assume that some trace j survives but does not contain u_L . Let $u_{\min} = \arg \min\{y_v : v \in C_i^{(d)}, t_v = j\}$ denote the first original $\mathbf{1}$ in this trace that belongs to $C_i^{(d)}$'s block (By definition $p_{u_{\min}} = p_{\min}(v, U)$ for each $v : t_v = j$). Then we know that

$$\begin{aligned} \text{len}(\tilde{x}_j^{(d,i)}) & \leq (p_{u_R} - p_{u_{\min}})/2 + \sqrt{(p_{u_R} - p_{u_{\min}}) \log(nmk)} \\ & \leq (p_{u_R} - p_{u_{\min}})/2 + \sqrt{2L^{(d,i)} \log(nmk)} \end{aligned}$$

but since $\tilde{x}_j^{(d,i)}$ passed through the length filter, we also have a lower bound on its length, and so we get that

$$p_{u_{\min}} - p_{u_L} \leq 4\sqrt{2L^{(d,i)} \log(nmk)} \leq 8\sqrt{k\tau_d \log(nmk)}$$

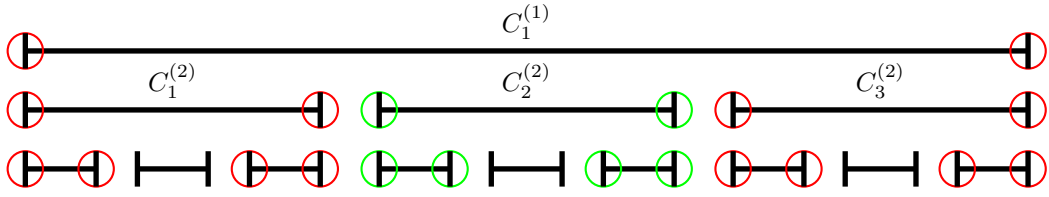
where the last inequality follows from Lemma 17. ◀

Proof of Lemma 16. The proof here follows the discussion in the previous subsection. Fix a component $C_i^{(d-1)}$ with corresponding block $U_i^{(d-1)} \subset [k]$ at level $d-1$ and assume that all three lemmas apply for all previous levels. For a subtrace $x_j^{(d-1,i)}$ in this component observe and recall the definition $u_{i,1}^{(d-1)} = \min\{y_v : v \in C_i^{(d-1)}\}$ and $p_{\min}(v, U_i^{(d-1)})$, which is the position of the first $\mathbf{1}$ in $U_i^{(d-1)}$ that appears in trace $t_v = j$. Since the length of the subtrace is at most $2k\tau_{d-1}$ by Lemma 17 we get that

$$\begin{aligned} & |z_v^{(d-1)} - (p_{t_v} - p_{u_{i,1}^{(d-1)}})/2| \\ & \leq |z_v^{(d-1)} - (p_{t_v} - p_{\min}(v, U_i^{(d-1)}))/2| + |p_{\min}(v, U_i^{(d-1)}) - p_{u_{i,1}^{(d-1)}}|/2 \\ & \leq \sqrt{2k\tau_{d-1} \log(mnk)} + 8\sqrt{k\tau_{d-1} \log(mkn)} = \tau_d/8. \end{aligned} \tag{2}$$

Here the last inequality uses Hoeffding's bound along with Lemma 18 at level $d-1$. This implies that the clustering at level d is consistent. ◀

Proof of Theorem 15. First take a union bound over $D \leq \log \log n$ applications of the three lemmas, so that the total failure probability is $cD/n^2 \leq 1/n$. From now, assume that the events in the three lemmas all hold for all levels. In particular, this implies that the components $C_i^{(D)}$ are consistent. We must verify that the clusters are pure and then track how many vertices remain.



■ **Figure 1** De-biasing of traces. The figure displays the regions of the original string x that correspond to each connected component found in the algorithm. The end-points of each component correspond to 1s in the original string. To de-bias the length filter for component $C_1^{(1)}$ at level 1, we identify and retain only the traces that contain *all* of the 1s colored red above. Then, to de-bias the length filter at $C_2^{(2)}$ at level 2, we identify and retain only the traces that contain *all* of the green 1s.

For the first claim, let us revisit the proof of Lemma 16. If two vertices, say v, v' , in a component at level $D - 1$ corresponded to different 1s, say u, u' then by the gap condition, we know that $|p_u - p_{u'}| \geq g$. On the other hand, we know that (2) holds, and we will use this to prove that no edge appears between these vertices. We have that

$$|z_v - z_{v'}| \geq \tau_D/8 + \tau_D/8 + |p_{t_v} - p_{t_{v'}}|/2 \geq \tau_D/4 + g/2,$$

and so, if $g/2 \geq \tau_D$, then the two vertices will not share an edge. The argument applies for all pairs and hence the clusters at level D are pure, which establishes the first claim in the Theorem 15.

For the second claim, note that by Lemma 18, for every component at every level, if a trace contains the two endpoints of that component, then it will survive the filter. Hence, in every filtering step we expect to retain 1/4 of the subtraces passing through, and, by a Chernoff bound, we will retain 1/5 of the subtraces except with $\exp(-\Omega(n))$, provided $m \geq n$. Since we perform $D = \log \log n$ levels, we retain $m/5^{\log \log n} = m/\log^5(n)$ traces in each cluster with high probability. ◀

Removing Bias: The reverse recursion

Now that we have isolated the vertices into pure clusters, we need to work our way up through the recursion to remove biases introduced by the hierarchical clustering. For any component $C_i^{(D-1)}$ corresponding to block $U_i^{(D-1)} \subset [k]$ at level $D - 1$, since the components at level D are pure, we can identify exactly the subtraces that contain the first and last 1 in the block. We throw away all other traces, which de-bias the length filter at level $D - 1$.

Unfortunately for a component $C_i^{(d-1)}$ corresponding to a block $U_i^{(d-1)}$ at level $d - 1$, we cannot identify exactly the subtraces that contain the exactly the first and last 1 in the block. However, we know that $C_i^{(d-1)}$ is further refined into sub-components $\{C_{i'}^{(d)}\}$ at level d , and by induction we can identify all the traces that contain the left-most and right-most 1 in the left-most and right-most sub-components. We identify all such traces and eliminate the rest to debias the length filter at level $d - 1$. See Figure 1 for an illustration.

To debias this length filter, we filter based on the presence of two 1s at level $d - 1$ (just the end points), and two further 1s at level d (the inner endpoints of the first and last sub-components), four further 1s at $d + 1$, and so on. So, just to debias the length filter at level $d - 1$ we require $2^{D-(d-1)}$ 1s to be present. Since we must debias all length filters above a particular component, we require the presence of $\sum_{d=1}^{D-1} 2^{D-d} \leq 2^D \leq \log_2(n)$ 1s. The probability of all $\log_2(n)$ of these 1s appearing is $1/n$ and by Chernoff bound, with high probability at least $m/2n$ of our traces will contain all of these 1s.

For any $1, u$, in the original string, let S denote the subset of $\log_2(n)$ 1s, whose presence we require to debias the length filters above the pure component containing u . After the debiasing step, the remaining vertices in the component containing u have z_v values distributed as

$$z_v \sim \text{Bin}(p_u - 1 - |S_L|, 1/2) + (|S_L| + 1)$$

where $|S_L|$ is the number of 1s in $|S|$ that appear before u in the sequence, and the final 1 is due to the presence of u . Using the binomial mean estimator, we can therefore estimate p_u with probability $1 - 1/n$, provided $m/n \geq n^2 \log(n)$. Thus, $\text{poly}(n)$ traces suffice to recover all p_u values, provided that $g > \tau_D$ and $D = \log_2 \log_2 n$. This proves Theorem 2.

4 Reconstructing Arbitrary Matrices

Recall that in the matrix reconstruction problem, we are given samples of a matrix $X \in \{0, 1\}^{\sqrt{n} \times \sqrt{n}}$ passed through a *matrix deletion channel*, which deletes each row and each column independently with probability $p = 1 - q$. In this section we prove Theorem 4.

► **Theorem** (Restatement of Theorem 4). *For matrix reconstruction, $\exp(O(n^{1/4} \sqrt{p \log n}/q))$ traces suffice to recover an arbitrary matrix $X \in \{0, 1\}^{\sqrt{n} \times \sqrt{n}}$, where p is the deletion probability and $q = 1 - p$.*

The bulk of the proof involves designing a procedure to test between two matrices X and Y . This test is based on identifying a particular received entry where the traces must differ significantly, and to show this, we analyze a certain bivariate Littlewood polynomial, which is the bulk of the proof. Equipped with this test, we can apply a union bound and simply search over all pairs of matrices to recover the string.

For a matrix $X \in \{0, 1\}^{\sqrt{n} \times \sqrt{n}}$, let \tilde{X} denote a matrix trace. Let us denote the $(i, j)^{\text{th}}$ entry of the matrix as $X_{i,j}$, $i, j = 0, 1, \dots, \sqrt{n} - 1$, an indexing protocol we adhere to for every matrix. For two complex numbers $w_1, w_2 \in \mathbb{C}$, observe that

$$\begin{aligned} \mathbb{E} \left[\sum_{i,j=0}^{\sqrt{n}-1} \tilde{X}_{i,j} w_1^i w_2^j \right] &= q^2 \sum_{i,j} w_1^i w_2^j \sum_{k_i \geq i, k_j \geq j} X_{k_i, k_j} \binom{k_i}{i} \binom{k_j}{j} p^{k_i-i} q^i p^{k_j-j} q^j \\ &= q^2 \sum_{k_1, k_2=0}^{\sqrt{n}-1} X_{k_1, k_2} (qw_1 + p)^{k_1} (qw_2 + p)^{k_2} \end{aligned}$$

Thus, for two matrices X, Y , we have

$$\begin{aligned} \frac{1}{q^2} \mathbb{E} \left[\sum_{i,j=0}^{\sqrt{n}-1} (\tilde{X}_{i,j} - \tilde{Y}_{i,j}) w_1^i w_2^j \right] &= \sum_{k_1, k_2=0}^{\sqrt{n}-1} (X_{k_1, k_2} - Y_{k_1, k_2}) (qw_1 + p)^{k_1} (qw_2 + p)^{k_2} \\ &\triangleq A(z_1, z_2) \end{aligned}$$

where we are rebinding $z_1 = qw_1 + p$ and $z_2 = qw_2 + p$. Observe that $A(z_1, z_2)$ is a *bivariate Littlewood polynomial*; all coefficients are in $\{-1, 0, 1\}$, and the degree is \sqrt{n} . For such polynomials, we have the following estimate, which extends a result due of Borwein and Erdélyi [6] for univariate polynomials.

► **Lemma 19.** *Let $f(z_1, z_2)$ be non-zero degree n Littlewood polynomial. Then,*

$$|f(z_1^*, z_2^*)| \geq \exp(-C_1 L^2 \log n)$$

for some $z_1^* = \exp(i\theta_1), z_2^* = \exp(i\theta_2)$ where $|\theta_1|, |\theta_2| \leq \pi/L$, and C_1 is a universal constant.

Proof. Fix $L > 0$ and define the polynomial

$$F(z_1, z_2) = \prod_{1 \leq a \leq L, 1 \leq b \leq L} f(z_1 e^{\pi i a/L}, z_2 e^{\pi i b/L}).$$

We first show that there exists z_1^*, z_2^* on the unit disk such that $F(z_1^*, z_2^*) \geq 1$. This follows from an iterated application of the maximum modulus principle. First factorize $F(z_1, z_2) = z_2^k G(z_1, z_2)$ where k is chosen such that $G(z_1, z_2)$ has no common factors of z_2 . Since F has non-zero coefficients, this implies that $G(z_1, 0)$ is a non-zero univariate polynomial. Further factorize $G(z_1, 0) = z_1^\ell H(z_1)$ so that terms in H have no common factors of z_1 . H is also a Littlewood polynomial and moreover it has non-zero leading term, so that $|H(0)| \geq 1$. Thus by the maximum modulus principle:

$$|F(z_1^*, z_2^*)| = |G(z_1^*, z_2^*)| \geq |G(z_1^*, 0)| \geq |H(z_1^*)| \geq |H(0)| \geq 1.$$

Now, for any $a, b \in \{1, \dots, L\}$ we have

$$1 \leq |F(z_1^*, z_2^*)| \leq |f((z_1^*)^{\pi i a/L}, (z_2^*)^{\pi i b/L})| \cdot n^{(L-1)},$$

where we are using the fact that $|f(z_1, z_2)| \leq n$. This proves the lemma, since we may choose a such that $(z_1^*)^{\pi i a/L} = \exp(i\theta)$ for $|\theta| \leq \pi/L$. \blacktriangleleft

Let $\gamma_L = \{e^{i\theta} : |\theta| \leq \pi/L\}$ denote the arc specified in Lemma 19. For any $z_1 \in \gamma_L$, Nazarov and Peres [26] provide the following estimate for the modulus of $w_1 = (z_1 - p)/q$:

$$\forall z \in \gamma_L : |(z - p)/q| \leq \exp(C_2 p/(Lq)^2).$$

Using these two estimates, we may sandwich $|A(z_1, z_2)|$ by

$$\exp(-C_1 L^2 \log n) \leq \max_{z_1, z_2 \in \gamma_L} |A(z_1, z_2)| \leq \frac{\exp(C' p \sqrt{n}/(Lq)^2)}{q^2} \sum_{ij} |\mathbb{E}[\tilde{X}_{ij} - \tilde{Y}_{ij}]|.$$

This implies that there exists some coordinate (i, j) such that

$$|\mathbb{E}[\tilde{X}_{ij} - \tilde{Y}_{ij}]| \geq \frac{q^2}{n} \exp\left(-C_1 L^2 \log n - \frac{C' p \sqrt{n}}{L^2 q^2}\right) \geq \frac{q^2}{n} \exp\left(-C \frac{n^{1/4} \sqrt{p \log n}}{q}\right),$$

where the second inequality follows by optimizing for L .

The remainder of the proof follows the argument of [26]: Since we have witnessed significant separation between the traces received from X and those received from Y , we can test between these cases with $\exp(O(n^{1/4} \sqrt{\log n}))$ samples (via a simple Chernoff bound). Since we do not know which of the 2^n traces is the truth, we actually test between all pairs, where the test has no guarantee if neither matrix is the truth. However, via a union bound, the true matrix will beat every other in these tests and this only introduces a poly(n) factor in the sample complexity.

5 Reconstructing Random Matrices

In this section, we prove Theorem 5: $O(\log n)$ traces suffice to reconstruct a random $\sqrt{n} \times \sqrt{n}$ matrix with high probability for any constant deletion probability $p < 1$. This is optimal since $\Omega(\log n)$ traces are necessary to just ensure that every bit appears in a least one trace.

Our result is proved in two steps. We first design an oracle that allows us to identify when two rows (or two columns) in different matrix traces correspond to the same row (resp. column) of the original matrix. We then use this oracle to identify which rows and columns of the original matrix have been deleted to generate each trace. This allows us to identify the original position of each bit in each trace. Hence, as long as each bit is preserved in at least one trace (and $O(\log n)$ traces is sufficient to ensure this with high probability), we can reconstruct the entire original matrix.

Oracle for Identifying Corresponding Rows/Columns. We will first design an oracle that given two strings t and t' distinguishes, for any constant $q > 0$, with high probability between the cases:

Case 1: t and t' are traces generated by the deletion channel with preservation probability q from the same random string $x \in_R \{0, 1\}^{\sqrt{n}}$

Case 2: t and t' are traces generated by the deletion channel with preservation probability q from independent random strings $x, y \in_R \{0, 1\}^{\sqrt{n}}$

If t and t' are two rows (or two columns) from two different matrix traces, then this test determines whether t and t' correspond to the same or different row (resp. column) of the original matrix. In Section 5.1, we show how to perform this test with failure probability at most $1/n^{10}$. In fact, the failure probability can be made exponentially small but a polynomially small failure probability will be sufficient for our purposes.

Using the Oracle for Reconstruction. Given $m = \Theta(\log n)$ traces we can ensure that every bit of X appears in at least one of the matrix traces with high probability. We then use this oracle to associate each row in each trace with the rows in other traces that are subsequences of the same original row. This requires at most $\binom{m\sqrt{n}}{2} \leq (m\sqrt{n})^2$ applications of the oracle and so, by the union bound, this can be performed with failure probability at most $(m\sqrt{n})^2/n^{10} \leq 1/n^8$ where the inequality applies for sufficiently large n .

After using the oracle to identify corresponding rows amongst the different traces we group all the rows of the traces into \sqrt{n} groups $G_1, \dots, G_{\sqrt{n}}$ where the expected size of each group is mq . We next infer which group corresponds to the i^{th} row of X for each $i \in [\sqrt{n}]$. Let f be the bijection between groups and $[\sqrt{n}]$ that we are trying to learn, i.e., $f(j) = i$ if the j^{th} group corresponds to the i^{th} row of X . It suffices to determine whether $f(j) < f(j')$ or $f(j) > f(j')$ for each pair $j \neq j'$. If there exists a matrix trace \tilde{X} that includes a row in G_j and a row in $G_{j'}$, then we can infer the relative ordering of $f(j)$ and $f(j')$ based on whether the row from G_j appears higher or lower in \tilde{X} than the row in $G_{j'}$. The probability there exists such a trace is $1 - (1 - q^2)^m \geq 1 - 1/\text{poly}(n)$ and we can learn the bijection f with high probability.

We also perform an analogous process with columns. After both rows and columns have been processed, we know exactly which rows and columns were deleted to form each trace, which reveals the original position of each received bit in each trace. Given that every bit of X appeared in at least some trace, this suffices to reconstruct X , proving Theorem 5.

► **Theorem** (Restatement of Theorem 5). *For any constant deletion probability $p < 1$, $O(\log n)$ traces are sufficient to reconstruct a random $X \in \{0, 1\}^{\sqrt{n} \times \sqrt{n}}$.*

5.1 Oracle: Testing whether two traces come from same random string

Define $S_i = \{2wi + j : j = 0, \dots, w - 1\}$ to be a contiguous subsets of size

$$w = 100n^{1/4} \sqrt{1/q \cdot \log n}.$$

Note that there are size w gaps between each S_i and S_{i+1} , i.e., w elements that are both larger than S_i and smaller than S_{i+1} . This will later help us argue that the bits in positions S_i and S_{i+1} in different traces are independent. Given a traces t, t' , define the three quantities: $X_i = \sum_{j \in S_i} t_j$, $Y_i = \sum_{j \in S_i} t'_j$ and $Z_i = (X_i - Y_i)^2$. We will show that by considering Z_0, Z_1, Z_2, \dots we can determine whether t and t' are traces of the same original string or traces of two different random strings.

68:22 Trace Reconstruction: Generalized and Parameterized

The basic idea is that if t and t' are generated by the same string, many of the bits summed to construct X_i and the bits summed to construct Y_i will correspond to the same bits of the original string; hence Z_i will be smaller than it would be if t and t' were generated from two independent random strings. To make this precise, we need to introduce some additional notation.

► **Definition 20.** For $A \subset \{0, 1, 2, \dots\}$, let $R_t(A)$ be the indices of the bits in the transmitted string that landed in positions A in trace t . Similarly define $R_{t'}(A)$. For example, if bits in position 0 and 2 were deleted during the transmission of t then $R_t(\{0, 1, 2\}) = \{1, 3, 4\}$.

The next lemma quantifies the overlap between $R_t(S_i)$ and $R_{t'}(S_i)$.

► **Lemma 21 (Deletion Patterns).** With high probability over the randomness of the deletion channel,

$$\forall i, |R_t(S_i) \cap R_{t'}(S_i)| \geq qw/2 \quad \text{and} \quad \forall i \neq j, |R_t(S_i) \cap R_{t'}(S_j)| = 0.$$

Note that conditioned on the second property, each Z_i is independent.

Proof. First note that by the Chernoff bound, for each $j \in [\sqrt{n}]$, the j^{th} bit of the original sequence appears in position $qj \pm r$ where $r = 5n^{1/4}\sqrt{q \log n}$ with high probability. The second part of the lemma follows since $r = wq/20 < w/20$ and therefore, with high probability, any bit in the original string will not appear in S_α in one trace and S_β in another for $\alpha \neq \beta$ because there was a size w gap between S_α and S_β .

For the first part of the lemma, for each S_i , define

$$S'_i = \{2wi/q + r/q, 2wi/q + r + 1, \dots, (2wi + w - 1)/q - r/q\}.$$

By the Chernoff Bound, with high probability the $w/q - 2r/q > 0.9w/q$ bits in S'_i positions in the original string arrive in positions S_i in the trace. Also with high probability, $0.9q^2|S'_i|$ of the bits in S'_i are transmitted in the generation of both t and t' . Hence, $|R_t(S_i) \cap R_{t'}(S_i)| \geq 0.9w/q \cdot 0.9q^2 > qw/2$ as required. ◀

We are now ready to argue that the values Z_0, Z_1, \dots are sufficient to determine whether or not t and t' are generated from the same random string.

► **Theorem 22.** Let $z_j = \sum_{i=0}^{g-1} Z_{jg+i}$ for $g = 96/q^2$ and $D = \text{median}(z_0, z_1, z_2, \dots, z_{\Theta(\log n)})$.
Case 1: If t and t' are generated from the same string, then $\Pr[D < (1 - q/4)gw/2] \geq 1 - 1/n^{10}$.

Case 2: If t and t' are generated from different strings, then $\Pr[D \geq (1 - q/4)gw/2] \geq 1 - 1/n^{10}$.

Proof. Throughout the proof we condition on the equations in Lemma 21 being satisfied. Note that this event is a function of the randomness of the deletion channel rather than the randomness of the strings being transmitted over the deletion channel.

First, suppose t and t' are generated from different strings. Then Z_i has the same distribution as the variable C in Lemma 23 when r is set to w . Hence, $\mathbb{E}[z_j] = gw/2$ and $\text{var}(z_j) \leq gw^2/2$. Therefore,

$$\Pr[z_j < (1 - q/4)gw/2] \leq \Pr[|z_j - \mathbb{E}[z_j]| \geq (q/4)gw/2] \leq \frac{\text{var}(z_j)}{\mathbb{E}[z_j]^2 \cdot q^2/16} \leq \frac{2}{qq^2/16} = 1/3.$$

Therefore, by the Chernoff bound, $D \geq (1 - q/4)gw/2$ with probability at least $1 - 1/n^{10}$.

Now, suppose t and t' are generated from the same string. Then, Z_i has the same distribution as C in Lemma 23 for some $r \leq w - qw/2$. Hence, $\mathbb{E}[z_j] = gr/2$ and $\text{var}(z_j) \leq gr^2/2$. Therefore,

$$\Pr[z_j \geq (1 - q/4)gw/2] \leq \Pr[|z_j - \mathbb{E}[z_j]| \geq (q/4)gw/2] \leq \frac{\text{var}(z_j)}{\mathbb{E}[z_j]^2 \cdot q^2/16} \leq \frac{2}{gq^2/16} = 1/3.$$

Therefore, by the Chernoff bound, $D < (1 - q/4)gw/2$ with probability at least $1 - 1/n^{10}$. ◀

► **Lemma 23.** *Let $A \sim \text{Bin}(h, 1/2)$ and $B \sim \text{Bin}(h, 1/2)$ be independent and $C = (A - B)^2$. Then,*

$$\mathbb{E}[C] = h/2 \quad \text{and} \quad \text{var}[C] \leq m^2/2.$$

Proof. The result follows by direct calculation:

$$\mathbb{E}[(A - B)^2] = \mathbb{E}[A^2] + \mathbb{E}[B^2] - 2\mathbb{E}[A]\mathbb{E}[B] = m(m + 1)/2 - m^2/2 = m/2$$

and

$$\text{var}((A - B)^2) = \mathbb{E}[(A - B)^4] - (m/2)^2 = m/2 + 6\binom{m}{2}/4 - m^2/4 = (2m - 1)m/4.$$

◀

6 Bounded Hamming Distance

In this section, we turn to the sparse testing problem. We show that is possible to distinguish between two strings x and y with Hamming distance $\Delta(x, y) < 2k$, given $\exp(O(k \log n))$ traces. This question is naturally related to sparse reconstruction, since the difference string $x - y \in \{-1, 0, 1\}^n$ is at most $2k$ sparse, but distinguishing two strings from traces is also at the core of our analysis in Section 2, as well as the analysis of Nazarov and Peres [26] and De et al. [11]. In particular given a testing routine, reconstruction simply requires applying the union bound.

In the binary symmetric channel (where each bit is flipped independently with some probability), distinguishing between two strings is easier if the Hamming distance is larger, since the two strings are farther apart. However, it is unclear if this intuition carries over to the deletion channel. In particular, the number of traces required for testing is unlikely to even be monotonic in the Hamming distance; if the Hamming distance is odd, then x and y have different Hamming weight, and we can estimate the Hamming weight using just $O(n)$ traces.

Our analysis uses a combinatorial result about k -decks due to Krasikov and Roditty [23], along with an approach first used in McGregor et al. [24].

► **Theorem 24** (Krasikov and Roditty [23]). *The k -deck of a string is the multi-set of length k subsequences. No two strings x, y of length n have the same k -deck if $\Delta(x, y) < 2k$.*

► **Theorem 25.** *The k -deck of a binary string can be determined exactly with $\exp(O(k \log n))$ traces from the symmetric deletion channel assuming $p \leq 1 - k/n$.*

Proof. We argue that sampling $\exp(O(k \log n))$ length k -subsequence of a string is sufficient to reconstruct the k -deck with high probability. The result then follows because if $p \leq 1 - k/n$, then with constant probability a trace generated by the deletion channel has length at least k and hence we can take a random k subsequence of such a trace as a random k subsequence from x .

Let f_u be the number of times that $u \in \{0, 1\}^k$ appears as a subsequence of x . Then, let X_u be the number of times u is generated if we sample $r = 3n^k \log n^k$ subsequences of length k uniformly at random. $\mathbb{E}[X_i] = rf_u/\binom{n}{k}$ and by an application of the Chernoff bound.

$$\Pr \left[|X_u \binom{n}{k} / r - f_u| \geq 1 \right] = \Pr \left[|X_u - \mathbb{E}[X]| \geq r / \binom{n}{k} \right] \leq \exp \left(-\frac{f_u^2 \cdot r f_u}{3 \binom{n}{k}} \right) \leq 1/n^k$$

where the last line follows given $f_u \geq 1$ (if $f_u = 0$ the bound is trivially true) and $r = 3n^k \log n^k$. Hence, by taking the union bound over all 2^k sequences u , it follows that we can determine the frequency of all length k subsequences with high probability. ◀

Theorem 3 follows directly from Theorem 24 and Theorem 25.

► **Theorem** (Restatement of Theorem 3). *For all $x, y \in \{0, 1\}^n$ such that $\Delta(x, y) < 2k$,*

$$m = \exp(O(k \log n))$$

traces are sufficient to be distinguished between x and y .

As noted earlier, if $\Delta(x, y)$ is odd then $\text{poly}(n)$ traces suffice. Also, regardless of the Hamming distance, if the location of the first and second positions (say i and j) where x and y differs by at least $\Omega(\sqrt{n \log n})$ then it is easy to show that expected weight of the length $i/2$ prefix of the traces differs by $\Omega(1/\text{poly}(n))$ and hence we can distinguish x and y with $\text{poly}(n)$ traces.

References

- 1 Dimitris Achlioptas and Frank McSherry. On spectral learning of mixtures of distributions. In *Conference on Learning Theory*, 2005.
- 2 Sanjeev Arora and Ravi Kannan. Learning mixtures of arbitrary gaussians. In *Symposium on Theory of Computing*, 2001.
- 3 Frank Ban, Xi Chen, Adam Frelich, Rocco A. Servedio, and Sandip Sinha. Beyond trace reconstruction: Population recovery from the deletion channel. *arXiv*, 2019. [arXiv:1904.05532](#).
- 4 Tugkan Batu, Sampath Kannan, Sanjeev Khanna, and Andrew McGregor. Reconstructing strings from random traces. In *Symposium on Discrete Algorithms*, 2004.
- 5 Mikhail Belkin and Kaushik Sinha. Polynomial learning of distribution families. In *Foundations of Computer Science*, 2010.
- 6 P. Borwein and T. Erdélyi. Littlewood-Type Problems on Subarcs of the Unit Circle. *Indiana University Mathematics Journal*, 1997.
- 7 Siu-On Chan, Ilias Diakonikolas, Rocco A Servedio, and Xiaorui Sun. Learning mixtures of structured distributions over discrete domains. In *Symposium on Discrete Algorithms*, 2013.
- 8 Mahdi Cheraghchi, Ryan Gabrys, Olgica Milenkovic, and João Ribeiro. Coded trace reconstruction. *arXiv e-prints*, page arXiv:1903.09992, March 2019. [arXiv:1903.09992](#).
- 9 Sanjoy Dasgupta. Learning mixtures of Gaussians. In *Foundations of Computer Science*, 1999.
- 10 Sami Davies, Miklos Z. Racz, and Cyrus Rashtchian. Reconstructing Trees from Traces. *arXiv e-prints*, page arXiv:1902.05101, February 2019. [arXiv:1902.05101](#).
- 11 Anindya De, Ryan O'Donnell, and Rocco A. Servedio. Optimal mean-based algorithms for trace reconstruction. In *Symposium on Theory of Computing*, 2017.
- 12 Jon Feldman, Ryan O'Donnell, and Rocco A Servedio. Learning mixtures of product distributions over discrete domains. *SIAM Journal on Computing*, 2008.
- 13 Anna C. Gilbert and Piotr Indyk. Sparse Recovery Using Sparse Matrices. *Proceedings of the IEEE*, 2010.

- 14 Moritz Hardt and Eric Price. Tight bounds for learning a mixture of two gaussians. In *Symposium on Theory of Computing*, 2015.
- 15 Lisa Hartung, Nina Holden, and Yuval Peres. Trace reconstruction with varying deletion probabilities. In *Workshop on Analytic Algorithmics and Combinatorics*, 2018.
- 16 Nina Holden and Russell Lyons. Lower bounds for trace reconstruction. *arXiv*, 2018. [arXiv: 1808.02336](https://arxiv.org/abs/1808.02336).
- 17 Nina Holden, Robin Pemantle, and Yuval Peres. Subpolynomial trace reconstruction for random strings and arbitrary deletion probability. In *Conference On Learning Theory, COLT 2018, Stockholm, Sweden, 6-9 July 2018.*, pages 1799–1840, 2018.
- 18 Thomas Holenstein, Michael Mitzenmacher, Rina Panigrahy, and Udi Wieder. Trace reconstruction with constant deletion probability and related results. In *Symposium on Discrete Algorithms*, 2008.
- 19 Samuel B Hopkins and Jerry Li. Mixture models, robustness, and sum of squares proofs. In *Symposium on Theory of Computing*, 2018.
- 20 Adam Tauman Kalai, Ankur Moitra, and Gregory Valiant. Efficiently learning mixtures of two Gaussians. In *Symposium on Theory of Computing*, 2010.
- 21 Sampath Kannan and Andrew McGregor. More on Reconstructing Strings from Random Traces: Insertions and Deletions. In *International Symposium on Information Theory*, 2005.
- 22 Géza Kós, Péter Ligeti, and Péter Sziklai. Reconstruction of matrices from submatrices. *Mathematics of Computation*, 2009. doi:10.1090/S0025-5718-09-02210-8.
- 23 I. Krasikov and Y. Roditty. On a Reconstruction Problem for Sequences. *Journal of Combinatorial Theory, Series A*, 1997.
- 24 Andrew McGregor, Eric Price, and Sofya Vorotnikova. Trace Reconstruction Revisited. In *European Symposium on Algorithms*, 2014.
- 25 Ankur Moitra and Gregory Valiant. Settling the polynomial learnability of mixtures of gaussians. In *Foundations of Computer Science*, 2010.
- 26 Fedor Nazarov and Yuval Peres. Trace reconstruction with $\exp(O(n^{1/3}))$ samples. In *Symposium on Theory of Computing*, 2017.
- 27 Yuval Peres and Alex Zhai. Average-Case Reconstruction for the Deletion Channel: Subpolynomially Many Traces Suffice. In *Symposium on Foundations of Computer Science*, 2017.
- 28 Krishnamurthy Viswanathan and Ram Swaminathan. Improved string reconstruction over insertion-deletion channels. In *Symposium on Discrete Algorithms*, 2008.

Generalized Assignment via Submodular Optimization with Reserved Capacity

Ariel Kulik

Computer Science Department, Technion, Haifa, Israel
kulik@cs.technion.ac.il

Kanthi Sarpatwar

IBM Research, Yorktown Heights, NY, USA
sarpatwa@us.ibm.com

Baruch Schieber

Department of Computer Science, New Jersey Institute of Technology, Newark, NJ, USA
sbar@njit.edu

Hadas Shachnai

Computer Science Department, Technion, Haifa, Israel
hadas@cs.technion.ac.il

Abstract

We study a variant of the *generalized assignment problem* (GAP) with group constraints. An instance of **Group GAP** is a set I of items, partitioned into L groups, and a set of m uniform (unit-sized) bins. Each item $i \in I$ has a size $s_i > 0$, and a profit $p_{i,j} \geq 0$ if packed in bin j . A group of items is *satisfied* if all of its items are packed. The goal is to find a feasible packing of a subset of the items in the bins such that the total profit from satisfied groups is maximized. We point to central applications of **Group GAP** in Video-on-Demand services, mobile Device-to-Device network caching and base station cooperation in 5G networks.

Our main result is a $\frac{1}{6}$ -approximation algorithm for **Group GAP** instances where the total size of each group is at most $\frac{m}{2}$. At the heart of our algorithm lies an interesting derivation of a submodular function from the classic LP formulation of **GAP**, which facilitates the construction of a high profit solution utilizing at most half the total bin capacity, while the other half is *reserved* for later use. In particular, we give an algorithm for submodular maximization subject to a knapsack constraint, which finds a solution of profit at least $\frac{1}{3}$ of the optimum, using at most half the knapsack capacity, under mild restrictions on element sizes. Our novel approach of submodular optimization subject to a knapsack *with reserved capacity* constraint may find applications in solving other group assignment problems.

2012 ACM Subject Classification Theory of computation \rightarrow Packing and covering problems; Theory of computation \rightarrow Submodular optimization and polymatroids; Mathematics of computing \rightarrow Linear programming; Mathematics of computing \rightarrow Approximation algorithms

Keywords and phrases Group Generalized Assignment Problem, Submodular Maximization, Knapsack Constraints, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.69

Related Version A full version of the paper is available at <https://arxiv.org/abs/1907.01745>.

Acknowledgements H. Shachnai's work was conducted during a visit to DIMACS partially supported by the National Science Foundation under grant number CCF-1445755.



© Ariel Kulik, Kanthi Sarpatwar, Baruch Schieber, and Hadas Shachnai;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 69; pp. 69:1–69:15



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

With the rapid adoption of cloud computing, wireless networks, and other modern platforms, resource allocation problems of various flavors have regained importance. One classic example is the *generalized assignment problem* (GAP). We are given a set of n items and m bins, $[m] = \{1, 2, \dots, m\}$. Each item $i \in [n]$ has a size $s_{i,j} > 0$ and a profit $p_{i,j} \geq 0$ when packed into bin $j \in [m]$. The goal is to feasibly pack in the bins a subset of the items of maximum total profit. GAP has been widely studied, with applications ranging from grouping and loading in manufacturing systems to land use optimization in regional planning (see, e.g., [2, 10]). In discrete optimization, GAP has received considerable attention also as a special case of the *separable assignment* problem and *submodular maximization* (see, e.g., [23, 14, 4, 5]). We consider a variant of GAP with group constraints. An instance of **Group GAP** consists of a set $I = \{1, 2, \dots, n\}$ of n items and m uniform (unit-sized) bins $M = \{1, \dots, m\}$. Each item $i \in I$ has a size $s_i > 0$ and a profit $p_{i,j} \geq 0$ when assigned to bin $j \in [m]$. The items in I are partitioned into $L \geq 1$ groups, $\mathcal{G} = \{G_1, \dots, G_L\}$. Given an assignment of items to bins, we say that a group is *satisfied* if all of its items are assigned. The goal is to find a feasible assignment of a subset of the items to bins such that the total profit from satisfied groups is maximized. Formally, a feasible assignment is a tuple (U_1, \dots, U_m) , such that $U_j \cap U_k = \emptyset$ for all $1 \leq j < k \leq m$, $U_j \subseteq I$ and $\sum_{i \in U_j} s_i \leq 1$, for all $1 \leq j \leq m$. Let $I(U) = \cup_j U_j$. Then, $G_\ell \in \mathcal{G}$ is satisfied if $G_\ell \subseteq I(U)$. Let $\mathcal{G}_s = \{G_{\ell_1}, \dots, G_{\ell_t}\}$ be the set of satisfied groups and $I(\mathcal{G}_s) = \cup_{G_\ell \in \mathcal{G}_s} G_\ell$. Then we seek an assignment (U_1, \dots, U_m) for which $\sum_{j=1}^m \sum_{i \in U_j \cap I(\mathcal{G}_s)} p_{i,j}$ is maximized.

The following scenario suggests a natural application for **Group GAP**. Consider a Video-on-Demand (VoD) service where each video is given as a collection of segments. The system has a set of m servers of uniform capacity distributed over multiple locations. To obtain revenue from a video the system must store all of its segments (possibly on different servers). The revenue from a specific video also depends on the servers which store the segments. This is due to the content delivery costs resulting from the distance between the servers and the predicted location of the video audience. The objective of the VoD service provider is to select a subset of segments and an allocation of these segments to servers so as to maximize the total revenue. In [16] we describe central applications of **Group GAP** in mobile Device-to-Device network caching and in base station cooperation in 5G networks.

1.1 Prior Work

We note that a **Group GAP** instance in which each group consists of a *single* item yields an instance of classic GAP where each item takes a single size across the bins, and all the bins have identical capacities. GAP is known to be APX-hard already in this case, even if there are only two possible item sizes, and each item can take one of two possible profits [8]. Thus, most of the previous research focused on obtaining efficient approximate solutions.¹ Fleischer et al. [14] obtained a $(1 - e^{-1})$ -approximation for GAP, as a special case of the *separable assignment problem*. Feige and Vondrák [12] obtained the current best known ratio of $1 - e^{-1} + \varepsilon$, for some absolute constant $\varepsilon > 0$.

¹ Given an algorithm \mathcal{A} , let $\mathcal{A}(I)$, $OPT(I)$ denote the profit of the solution output by \mathcal{A} and by an optimal solution for a problem instance I , respectively. For $\rho \in (0, 1]$, we say that \mathcal{A} is a ρ -approximation algorithm if, for any instance I , $\frac{\mathcal{A}(I)}{OPT(I)} \geq \rho$.

Chen and Zhang [9] studied the problem of *group packing of items into multiple knapsacks* (GMKP), a special case of Group GAP where the profit of each item is the same across the bins. Let $\text{GMKP}(\delta)$ be the restriction of GMPK to instances in which the total size of items in each group is at most δm (that is, a factor δ of the total capacity of all bins). For $\delta > \frac{2}{3}$, the paper [9] rules out the existence of a constant factor approximation for $\text{GMKP}(\delta)$, unless $\text{P} = \text{NP}$. For $\frac{1}{3} < \delta \leq \frac{2}{3}$, the authors show that there is no $(\frac{1}{2} + \varepsilon)$ -approximation for $\text{GMKP}(\delta)$, unless $\text{P} = \text{NP}$, and derive a nearly matching $(\frac{1}{2} - \varepsilon)$ -approximation, for any $\varepsilon > 0$. The paper presents also approximation algorithms and hardness results for other special cases of GMPK.

There has been earlier work also on variants of Group GAP with the added constraint that in any feasible assignment there is at most one item from G_ℓ in bin j , for any $1 \leq \ell \leq L$, $j \in [m]$. Adany et al. [1] considered this problem, called *all-or-nothing* GAP (AGAP). They presented a $(\frac{1}{19} - \varepsilon)$ -approximation algorithm for general instances, and a $(\frac{1}{3} - \varepsilon)$ -approximation for the special case where the profit of an item is identical across the bins, called the *group packing* (GP) problem. Sarpatwar et al. [20] consider a more general setting for AGAP, where each group of items is associated with a time window in which it can be packed. The paper shows that this variant of the problem, called χ -AGAP, admits an $\Omega(1)$ -approximation, assuming the time windows are large enough relative to group sizes. Specifically, for a group G_ℓ having a time window of m slots ($= m$ bins), it is assumed that $s(G_\ell) \leq \frac{m}{20}$.

1.1.1 Submodular Maximization

Given a finite set Ω , a function $f : 2^\Omega \rightarrow \mathbb{R}$ is *submodular* if for every $S, T \subseteq \Omega$ we have

$$f(S) + f(T) \geq f(S \cup T) + f(S \cap T).$$

An equivalent definition of submodularity refers to its diminishing returns: for any $T \subseteq S \subseteq \Omega$, and $u \in \Omega \setminus S$,

$$f(S \cup \{u\}) - f(S) \leq f(T \cup \{u\}) - f(T).$$

A set function f is *monotone* if for every $S \subseteq T \subseteq \Omega$ it holds that $f(S) \leq f(T)$. Submodular functions arise naturally in a wide variety of optimization problems, ranging from coverage problems and graph cut problems to welfare problems (see [6] for a survey on submodular functions). Submodular optimization under various constraints has been widely studied in the past four decades (see, e.g., [22, 7, 13] and [6] and the references therein).

The problem of maximizing a monotone submodular function subject to a knapsack constraint is defined as follows. We are given an oracle to a monotone, non-negative submodular function $f : 2^\Omega \rightarrow \mathbb{R}_{\geq 0}$. Each element $i \in \Omega$ is associated with a size $s_i \geq 0$. We are also given a capacity $B > 0$. The objective is to find a subset $S \subseteq \Omega$ such that $\sum_{i \in S} s_i \leq B$ and $f(S)$ is maximized. The best known result is a $(1 - e^{-1})$ -approximation algorithm due to Sviridenko [22]. The ratio of $(1 - e^{-1})$ cannot be improved even when f is a coverage function and element sizes are uniform, unless $\text{P} = \text{NP}$ [11]. A matching lower bound of $(1 - e^{-1})$ is known also for the oracle model with no complexity assumption [19].

1.2 Contribution and Techniques

Our main result is a $\frac{1}{6}$ -approximation algorithm for Group GAP instances where the total size of each group is at most $\frac{m}{2}$. We note that when group sizes can be arbitrary in $(0, m]$, Group GAP cannot be approximated within any bounded ratio, even if item profits are identical

across the bins, and $m = 2$, unless $P = NP$. Indeed, in this case, deciding whether a single group of items of total size 2 and total profit 1 can be packed in the bins yields an instance of PARTITION, which is NP-complete [15]. Furthermore, even if group sizes are restricted to be no greater than δm , for some $\delta > \frac{2}{3}$, then Group GAP still cannot be approximated within a constant factor, as it generalizes $\text{GMKP}(\delta)$, for which the paper [9] shows hardness of approximation. Similarly, as we consider in this paper a generalization of $\text{GMKP}(\frac{1}{2})$, it follows from [9] that our problem cannot be approximated within ratio better than $\frac{1}{2}$.

In solving Group GAP we combine the framework of Adany et al. [1] with the rounding technique of Shmoys and Tardos [21]. The framework of [1] uses submodular maximization to select a collection of groups for the solution. It then finds a feasible assignment for the selected groups.

At the heart of our algorithm lies an interesting derivation of a submodular function from the classic LP formulation of GAP, which facilitates the construction of a high profit solution utilizing at most half the total bin capacity. In particular, we give an algorithm for submodular maximization subject to a knapsack constraint, which finds a solution occupying at most half the knapsack capacity, while the other half is *reserved* for later use.² We show that this algorithm achieves an approximation ratio of $\frac{1}{3}$ relative to an optimal solution that may use the whole knapsack capacity. We note that this ratio is tight. Indeed, it is easy to construct an instance for which the best solution with half the knapsack capacity has only $\frac{1}{3}$ the profit of the optimal solution with full knapsack capacity. We also note that a naive application of the algorithm of Sviridenko [22] with half the knapsack capacity will only guarantee a $\frac{1-e^{-1}}{3} \approx \frac{1}{4.7}$ -approximation.

To obtain an integral solution, given a fractional assignment of the selected groups, we apply the rounding technique of Shmoys and Tardos [21], followed by a filling phase. We show that if the total size of the items in the selected groups is at most $\frac{m}{2}$, the rounding procedure yields a *feasible* assignment of the selected groups, whose profit is at least half the value of the submodular function. Our novel approach of submodular optimization subject to a knapsack *with reserved capacity* constraint may find applications in solving other group assignment problems.

2 Approximation Algorithm

In this section we present an approximation algorithm for Group GAP. We first introduce several definitions and tools that will be used as building blocks of our algorithm.

2.1 Basic Definitions and Tools

2.1.1 The Submodular Relaxation

For simplicity, we assume that all the numbers are rational. For a subset of elements $I' \subseteq I$, let $s(I') = \sum_{i \in I'} s_i$ be the total size of the elements in I' . We assume throughout the discussion that every $G_\ell \in \mathcal{G}$ satisfies $s(G_\ell) \leq \frac{m}{2}$. We define below a function $\phi : 2^I \rightarrow \mathbb{R}_{\geq 0}$. Let $x_{i,j} \in \{0,1\}$ be an indicator for the assignment of item i to bin j , for $1 \leq i \leq n$, $1 \leq j \leq m$. The following is a linear program associated with a subset $S \subseteq I$, in which $x_{i,j} \geq 0$ for all i, j .

² We assume throughout the discussion that the size of each element is at most half the knapsack capacity.

$$\begin{aligned}
LP(S): \text{ maximize } & \sum_{i \in I, j \in M} x_{i,j} \cdot p_{i,j} \\
\text{subject to: } & \sum_{j \in M} x_{i,j} \leq 1 \quad \forall i \in I \tag{1} \\
& \sum_{i \in I} x_{i,j} \cdot s_i \leq 1 \quad \forall j \in M \tag{2} \\
& x_{i,j} = 0 \quad \forall i \in I \setminus S, j \in M \\
& x_{i,j} \geq 0 \quad \forall i \in I, j \in M
\end{aligned}$$

Note that, by the above constraints, all solutions for the LP have the same dimension, regardless of the size of S . We define $\phi(S)$ as the optimal value of $LP(S)$.

We denote the profit of a solution x for the linear program by $p \cdot x = \sum_{i \in I, j \in M} x_{i,j} \cdot p_{i,j}$. In Section 3 we prove the next result.

► **Theorem 1.** *The function ϕ is submodular.*

We note that ϕ is also monotone and non-negative. We use ϕ to define the *group function* $\psi : 2^{\mathcal{G}} \rightarrow \mathbb{R}_{\geq 0}$. For any $G^* \subseteq \mathcal{G}$ let $I(G^*) = \bigcup_{G_\ell \in G^*} G_\ell$ and $\psi(G^*) = \phi(I(G^*))$. As ϕ is submodular, monotone and non-negative, it is easy to see that ψ is submodular, monotone and non-negative as well. We optimize ψ subject to a knapsack (budget) constraint, using the next general result.

► **Theorem 2** (Submodular optimization with reserved capacity). *Let $\Omega = \{1, \dots, n\}$ be a ground set, and $m \geq 0$ a knapsack capacity. Each $i \in \Omega$ is associated with non-negative size $s_i \leq \frac{m}{2}$. Let $f : 2^\Omega \rightarrow \mathbb{R}_{\geq 0}$ be a non-negative monotone submodular function, and $OPT = \max\{f(S) \mid S \subseteq \Omega, \sum_{i \in S} s_i \leq m\}$. Then Algorithm 2 (in Section 4) finds in polynomial time³ a subset $S \subseteq \Omega$ satisfying $f(S) \geq \frac{OPT}{3}$ and $\sum_{i \in S} s_i \leq \frac{m}{2}$.*

The proof of Theorem 2 is given in Section 4.

2.1.2 Solution Types

Our algorithm uses a few types of intermediate solutions for **Group GAP**, as defined below. Given $G^* \subseteq \mathcal{G}$, we say that a solution x for $LP(I(G^*))$ is a *fractional solution*. Let $U = (U_1, \dots, U_m)$ be an assignment of elements to bins, where U_j is the set of elements assigned to bin j . Then $I(U) = \bigcup_{j=1}^m U_j$ is the subset of elements packed in the bins. We say that U is *feasible* if for each bin $1 \leq j \leq m$ we have $s(U_j) \leq 1$. We say that U is *almost feasible* if for each bin $1 \leq j \leq m$ there is an element u_j^* such that $s(U_j \setminus \{u_j^*\}) \leq 1$. We also define the profit of an assignment as $p(U) = \sum_{1 \leq j \leq m} \sum_{i \in U_j} p_{i,j}$.

Our algorithm first obtains a fractional solution, which is then converted to an almost feasible solution. Finally, the algorithm converts this solution to a feasible one. We now state the results used in these conversion steps.

³ The explicit representation of a submodular function might be exponential in the size of its ground set. Thus, it is standard practice to assume that the function is accessed via a value oracle. Then the number of operations and oracle calls is polynomial in the size of Ω and the maximum length of the representation of $f(S)$.

► **Theorem 3.** *Given $G^* \subseteq \mathcal{G}$, such that $s(I(G^*)) \leq m$, and a fractional solution x for $LP(I(G^*))$, it is possible to construct in polynomial time an almost feasible assignment U such that $p(U) \geq p \cdot x$, and $I(U) = I(G^*)$.*

The theorem easily follows by applying a rounding technique of [21] to a fractional solution in which every element in $I(G^*)$ is fully assigned (fractionally, in multiple bins). We note that such a solution always exists, since $s(I(G^*)) \leq m$. We give the proof in the full version of the paper [16]. To convert an almost feasible solution to a feasible one we use the following result (we give the proof in Section 5).

► **Theorem 4.** *Let $U = (U_1, \dots, U_m)$ be an almost feasible assignment such that $s(I(U)) \leq \frac{m}{2}$, then U can be converted in polynomial time to a feasible assignment U' , with $I(U') = I(U)$ and $p(U') \geq \frac{1}{2}p(U)$.*

2.2 The Algorithm

Our approximation algorithm for Group GAP follows easily from the tools presented in Section 2.1. Initially, we solve the problem of maximizing a submodular function subject to a knapsack with reserved capacity constraint for the set function ψ . Then we solve the linear program and convert the solution to a feasible assignment. We give the pseudocode in Algorithm 1.

■ Algorithm 1 Group GAP Algorithm.

-
- 1: Solve the submodular optimization problem: $\max_{\{S \subseteq \mathcal{G}, \sum_{G_\ell \in S} s(G_\ell) \leq m/2\}} \psi(S)$ using Algorithm 2. Let S^* be the solution found.
 - 2: Find a (fractional) solution x for $LP(I(S^*))$ that realizes $\psi(S^*)$.
 - 3: Use Theorem 3 to convert x to an almost feasible assignment U with $I(U) = I(S^*)$.
 - 4: Use Theorem 4 to convert U into a feasible solution; return this solution.
-

► **Theorem 5.** *Algorithm 1 is a polynomial time $\frac{1}{6}$ -approximation algorithm for Group GAP when the total size of a single group is bounded by $\frac{m}{2}$; that is, $\forall G_\ell \in \mathcal{G} : \sum_{i \in G_\ell} s_i \leq \frac{m}{2}$.*

Proof. It is easy to see that the algorithm runs in polynomial time. By Theorem 2, we have that $\psi(S^*) \geq \text{OPT}/3$, where OPT is the value of the optimal solution for the original instance.

By Theorems 3 and 4, we are guaranteed to find in Steps 3–4 a feasible assignment U of all elements in $I(S^*)$, such that $p(U) \geq \frac{1}{2}\psi(S^*) \geq \frac{1}{6}\text{OPT}$. ◀

3 Submodularity

In this section we show that the function ϕ is submodular. Our proof builds on the useful relation of our problem to maximum weight bipartite matching. Let $G = (A \cup B, E)$ be a bipartite (edge) weighted graph, where $|B| \geq |A|$. Assume that the graph is complete (by adding zero weight edges if needed). For $e \in E$, let $W(e)$ be the weight of edge e , and for $F \subseteq E$, let $W(F) = \sum_{e \in F} W(e)$ be the total weight of edges in F . For $S \subseteq A$, define $h(S)$ to be the value of the maximum weight matching in $G[S \cup B]$, the graph induced by $S \cup B$. We call h the *partial maximum weight matching function*. The next result was shown by Bar-Noy and Rabanca [3].

► **Theorem 6.** *If the edge weights are non-negative then the function h is (monotone) submodular.*

We give a simpler proof in the full version of the paper [16]. We are now ready to prove our main result.

Proof of Theorem 1. We first note that since all numbers are rational, for some $N \in \mathbb{Z}^+$, we can write $s_i = \frac{\hat{s}_i}{N}$, where $\hat{s}_i \in \mathbb{Z}^+$ for all $i \in I$.⁴

Now, set the capacity of each bin $1 \leq j \leq m$ to be $b_j = N$, and let $0 \leq y_{i,j} \leq \hat{s}_i$ indicate the size of item i assigned to bin j . For a subset of items $S \subseteq I$, we now write the following linear program.

$$M(S): \text{ maximize } \sum_{i \in I} \frac{1}{\hat{s}_i} \sum_{j \in M} y_{i,j} \cdot p_{i,j}$$

$$\text{ subject to: } \sum_{j \in M} y_{i,j} \leq \hat{s}_i \quad \forall i \in I \quad (3)$$

$$\sum_{i \in I} y_{i,j} \leq N \quad \forall j \in M \quad (4)$$

$$y_{i,j} = 0 \quad \forall i \in I \setminus S, j \in M$$

$$y_{i,j} \geq 0 \quad \forall i \in I, j \in M$$

Indeed, Constraint (3) ensures that the total size assigned for item i over the bins is upper bounded by \hat{s}_i , and Constraint (4) guarantees that the capacity constraint is satisfied for all the bins $j \in M$. Given a subset of elements $S \subseteq I$, let $\eta(S)$ be the value of an optimal solution for $M(S)$.

Now, observe that any feasible solution for $LP(S)$ induces a feasible solution for $M(S)$ of the same value, by setting $y_{i,j} = x_{i,j} \cdot \hat{s}_i$ for all $i \in I$ and $j \in M$. Similarly, a feasible solution for $M(S)$ induces a feasible solution for $LP(S)$ of the same value. Hence, $\phi(S) = \eta(S)$ for all $S \subseteq I$.

By the above discussion, to prove the theorem it suffices to show that η is submodular. Given our Group GAP instance, we construct the following bipartite graph G . For each item $i \in I$, we define \hat{s}_i vertices, $V_i = \{v_{i,1}, \dots, v_{i,\hat{s}_i}\}$. For each bin $j \in M$, we define N vertices $U_j = \{u_{j,1}, \dots, u_{j,N}\}$. For any $i \in [n]$ and $j \in [m]$, there are edges $(v_{i,s}, u_{j,r})$ of weight $p_{i,j}/\hat{s}_i$, for all $1 \leq s \leq \hat{s}_i$, $1 \leq r \leq N$. Let $V_I = \cup_{i \in I} V_i$, $U_M = \cup_{j \in M} U_j$, and let E be the set of edges. Consider the bipartite graph $G = (V_I \cup U_M, E)$. W.l.o.g we may assume that $|U_M| \geq |V_I|$; otherwise, we can add new bins $j = m+1, m+2, \dots$ with corresponding sets of vertices $U_j = \{u_{j,1}, \dots, u_{j,N}\}$ and zero weight edges $(v_{i,s}, u_{j,r})$ for all $i \in [n]$, $1 \leq s \leq \hat{s}_i$, $1 \leq r \leq N$.

We note that, given a subset of items $S \subseteq I$, $M(S)$ is the linear programming relaxation of the problem of finding a maximum weight matching in the subgraph $G[V_S \cup U_M]$, where $V_S \subseteq V_I$ is the subset of vertices in G that corresponds to S . Using standard techniques (see, e.g., [17]), it can be shown that $M(S)$ has an optimal integral solution. Hence, $\eta(S) = h(V_S)$, where $h : 2^{V_I} \rightarrow \mathbb{R}_{\geq 0}$ is a partial maximum weight matching function in G . By Theorem 6, h is (monotone) submodular. Hence, η is also (monotone) submodular. ◀

⁴ Note that N , which may be arbitrarily large, is used just for the proof. Our algorithm does not rely on obtaining a solution (or an explicit formulation) for $M(S)$.

4 Submodular Optimization with Reserved Capacity

In this section we prove Theorem 2. We start with some definitions and notation. Assume we are given a ground set $\Omega = \{1, \dots, n\}$ and capacity $m > 0$, where each element $i \in \Omega$ is associated with a non-negative size $s_i \leq \frac{m}{2}$. For $S \subseteq \Omega$, let $s(S) = \sum_{i \in S} s_i$. Also, for $S, T \subseteq \Omega$ let $f_S(T) = f(S \cup T) - f(S)$. We use throughout this section basic properties of monotone submodular functions (see, e.g., [6]).

Algorithm 2 SUBMODULAROPT.

Input: A monotone submodular function $f : 2^\Omega \rightarrow \mathbb{R}_{\geq 0}$, sizes $s_i \geq 0$ for all $i \in \Omega$, and capacity $m > 0$.

Output: A subset of elements $R \subseteq \Omega$ such that $s(R) \leq \frac{m}{2}$.

```

1: procedure GREEDY( $g, m'$ )
2:   Set  $S = \emptyset, E = \Omega$ .
3:   while  $E \setminus S \neq \emptyset$  do
4:     Find  $i = \arg \max_{i \in E \setminus S} \frac{g_S(\{i\})}{s_i}$ 
5:     if  $s(S) + s_i \leq m'$  then set  $S = S \cup \{i\}$ .
6:     end if
7:     Set  $E = E \setminus \{i\}$ .
8:   end while
9:   Return  $S$ 
10: end procedure
11: Set  $R = \emptyset$ 
12: for every set  $S_e \subseteq \Omega, |S_e| \leq 6$  do
13:   for every set  $B \subseteq S_e, s(B) \leq m/2$  do
14:      $T = \text{GREEDY}(f_{S_e}, m/2 - s(B))$ 
15:     if  $f(B \cup T) \geq f(R)$  then Set  $R = B \cup T$ .
16:   end if
17: end for
18: end for
19: Return  $R$ 

```

In the following we give an outline of an algorithm for maximizing a monotone submodular function f subject to a knapsack *with reserved capacity* constraint. Specifically, assuming that the knapsack capacity is m for some $m > 0$, the algorithm solves the problem $\max_{\{S \subseteq \Omega: s(S) \leq \frac{m}{2}\}} f(S)$. The algorithm, SUBMODULAROPT, initially guesses the set of at most six items of highest profits in some optimal solution (for the problem with knapsack capacity m), and a subset of these profitable items, whose total size is at most $m/2$. Then the algorithm calls a procedure which applies the Greedy approach as in [22] to find the remaining items in the solution. We give a pseudocode of SUBMODULAROPT in Algorithm 2. It is important to note that while the algorithm produces a solution of size at most $m/2$, the analysis compares this solution against an optimal solution of size at most m .

The next lemma, which plays a key role in our analysis, follows from the technique presented in [22].

► **Lemma 7.** *Given the knapsack capacity $m > 0$, let $0 < m' \leq m^* \leq m$. Let $S^* \subseteq \Omega$ be a non-empty subset of elements, such that $s(S^*) \leq m^*$. Also, let $g : 2^\Omega \rightarrow \mathbb{R}$ be a monotone submodular function satisfying $g(\emptyset) = 0$, and let $S = \text{GREEDY}(g, m')$. Then, there is an element $i^* \in S^*$ such that $g(S) + g(\{i^*\}) \geq (1 - e^{-m'/m^*})g(S^*)$.*

In the full version of the paper [16] we prove a more general result (see Lemma 12 therein). Lemma 7 is obtained by setting $T = \emptyset$ in Lemma 12 in [16].

Lemma 7 was applied in [22] in the special case where $m' = m^*$. It was applied in conjunction with a guessing phase, used to ensure that the three most profitable elements in an optimal solution are selected by the algorithm, thus bounding the value of $g(\{i^*\})$.

Several difficulties arise while attempting to apply a similar approach to the problem with *reserved* capacity. The first one is that the most profitable elements in *any* optimal solution may already exceed the reduced capacity, and therefore cannot be added to the solution. Another difficulty is that even if these elements do fit in the smaller knapsack, one can easily come up with a scenario in which it is better not to include them in the solution.

To overcome these difficulties we use the following main observation. Given P_k , the set of $k = 6$ most profitable elements in an optimal solution⁵, and a partition of this set into two subsets B_1, B_2 , each of size at most $m/2$ (if such a partition exists), adding elements to either B_1 or B_2 using the greedy procedure leads to a solution of value at least one third of the value of an optimal solution. This observation comes into play in Case 2.2 in the proof of Theorem 2. The next technical lemma is used to prove this observation (we give the proof below).

► **Lemma 8.** *For $k = 6$, $p_A, p_B, S_A, S_B \geq 0$ such that $p_A + p_B \leq 1$ and $S_A + S_B \leq 1$, define*

$$h(p_A, p_B, S_A, S_B) = p_A + (1 - p_A - p_B) \left(1 - e^{-\frac{\frac{1}{2} - S_A}{1 - S_A - S_B}} \right) - \frac{p_A + p_B}{k}.$$

Then for p_1, p_2, S_1, S_2 such that $0 \leq p_1, p_2 \leq \frac{1}{3}$ and $0 \leq S_1, S_2 \leq \frac{1}{2}$ it holds that

$$\max(h(p_1, p_2, S_1, S_2), h(p_2, p_1, S_2, S_1)) \geq \frac{1}{3}.$$

Another main tool used in the proof of Theorem 2 is a simple partitioning procedure. It shows that P_k can either be partitioned into two sets as required in the above observation, or we reach a simple corner case (Case 2.1 in the proof) in which at least one third of the optimal value can be easily attained. For the latter case, we use the following result, due to [18].

► **Lemma 9.** *Let $g : 2^\Omega \rightarrow \mathbb{R}$ be a non-negative and monotone submodular function. Let $OPT = \max\{g(S) \mid S \subseteq \Omega, \sum_{i \in S} s_i \leq m\}$, and $S^* = \text{GREEDY}(g, m)$. Then either $g(S^*) \geq (1 - e^{-1/2})OPT$, or there is an element $i \in \Omega$ such that $g(\{i\}) \geq (1 - e^{-1/2})OPT$ and $s_i \leq m$.*

Proof of Theorem 2. (Submodular optimization with reserved capacity). It is easy to see that the running time of the algorithm is polynomial. Let $S \subseteq \Omega$, $s(S) \leq m$, $f(S) = OPT$, and $k = 6$.

Case 1: We first handle the case where $|S| < k$. We prove that in this case the algorithm finds a set R such that $f(R) \geq OPT/3$. Start with $A_1 = \emptyset$, iterate over the elements of S and add them to A_1 , as long as $s(A_1) \leq m/2$. If $S \neq A_1$, let $j \in S \setminus A_1$, and set $A_2 = \{j\}$ and $A_3 = S \setminus (A_1 \cup A_2)$. Clearly, $s(A_2) \leq m/2$, and since $s(A_1 \cup A_2) > m/2$ and $s(S) \leq m$, we have that $s(A_3) \leq m/2$. If $S = A_1$ set $A_2 = A_3 = \emptyset$.

⁵ The value $k = 6$ is derived from Lemma 8. It may be possible to obtain the same approximation ratio using smaller values of k , leading to a more efficient algorithm.

69:10 Generalized Assignment via Submodular Optimization

By the submodularity of f , we have $f(S) \leq f(A_1) + f(A_2) + f(A_3)$. Hence, for some $r \in \{1, 2, 3\}$, $f(A_r) \geq f(S)/3 = \text{OPT}/3$. We also have that $|A_r| \leq 5$; therefore, at some iteration of the algorithm $S_e = B = A_r$, and following this iteration $f(R) \geq \text{OPT}/3$.

Case 2: Assume now that $|S| \geq k$. Let $S = \{i_1, i_2, \dots, i_\ell\}$ such that the elements are ordered by their marginal profits: $i_j = \arg \max_{j \leq r \leq \ell} f_{\{i_1, \dots, i_{j-1}\}}(\{i_r\})$. Set $P_k = \{i_1, i_2, \dots, i_k\}$. Consider the following process. Start with $B_1 = \emptyset$ and $B_2 = \emptyset$. Iterate over the elements $i \in P_k$ in decreasing order by size. For each element i , let $t = \arg \min_{j=1,2} s(B_j)$. If $s(B_t) + s_i \leq m/2$ then $B_t = B_t \cup \{i\}$; otherwise, Stop. We now distinguish between two sub-cases for the termination of the process.

Case 2.1: Suppose that the process terminates due to an element i which cannot be added to any of the sets. Let B_1 and B_2 be the sets in this iteration. Also, set $B_3 = \{i\}$, $U = B_1 \cup B_2 \cup B_3$, and $L = S \setminus U$. W.l.o.g assume that $s(B_1) \geq s(B_2)$. As the process terminated, we have that $s(B_3) + s(B_2) = s_i + s(B_2) > m/2$. The sets B_1, B_2, B_3 and L form a partition of S , and $s(S) \leq m$. We conclude that $s(B_1) + s(L) \leq m/2$. Hence, $s(B_2) + s(L) \leq m/2$, and $s(B_3) + s(L) \leq m/2$ as well (it is easy to see that $s(B_3) \leq s(B_1)$). By the submodularity of f , $f(U) \leq f(B_1) + f(B_2) + f(B_3)$; thus, there is $j \in \{1, 2, 3\}$ such that $f(B_j) \geq f(U)/3$. As none of the sets B_1, B_2, B_3 is empty, we have that $|B_j| \leq |P_k| - 2 = 4$.

Let $T = \text{GREEDY}(f_U, m/2 - s(B_j))$. By Lemma 9, either

$$f_U(T) \geq (1 - e^{-1/2})f_U(L) \geq f_U(L)/3,$$

or there is $i \in L$ such that

$$f_U(\{i\}) \geq (1 - e^{-1/2})f_U(L) \geq f_U(L)/3.$$

In the former case, we can consider the iteration in which $S_e = U, B = B_j$. In this iteration, we have

$$f(B \cup T) \geq f(B_j) + f_U(T) \geq \frac{1}{3}(f(U) + f_U(L)) = \frac{1}{3}\text{OPT}.$$

In the latter case, we can consider the iteration where $S_e = B = B_j \cup \{i\}$, and in which

$$f(B \cup T) \geq f(B) \geq f(B_j) + f_{B_j}(\{i\}) \geq f(B_j) + f_U(\{i\}) \geq \frac{1}{3}\text{OPT}.$$

Case 2.2: The process terminated with B_1, B_2 satisfying $B_1 \cup B_2 = P_k$, and $s(B_1), s(B_2) \leq m/2$.

Let $p_1 = f(B_1)/\text{OPT}$, $p_2 = (f(P_k) - f(B_1))/\text{OPT}$, $S_1 = s(B_1)$, $S_2 = s(B_2)$ and $L = S \setminus P_k$. If $p_1 \geq \frac{1}{3}$ (or $p_2 \geq \frac{1}{3}$) we have that in the iteration where $S_e = P_k$ and $B = B_1$ (or $B = B_2$) the algorithm finds a solution of value at least $\text{OPT}/3$, and the theorem holds. Thus, we may assume that $p_1, p_2 \leq \frac{1}{3}$.

For $j = 1, 2$, let $T_j = \text{GREEDY}(f_{P_k}, m/2 - S_j)$. Using Lemma 7 with $S^* = L$, we have that there is $i_j \in L$ for which

$$f_{P_k}(T_j) + f_{P_k}(\{i_j\}) \geq (1 - e^{-\frac{\frac{1}{2}-S_j}{1-S_1-S_2}})f_{P_k}(L).$$

By the selection of elements in P_k , we have $f_{P_k}(\{i_j\}) \leq \frac{1}{k}f(P_k)$. Thus,

$$f_{B_j}(T_j) \geq f_{P_k}(T_j) \geq (1 - e^{-\frac{\frac{1}{2}-S_j}{1-S_1-S_2}})f_{P_k}(L) - \frac{1}{k}f(P_k).$$

Hence, in the iteration where $S_e = P_k, B = B_j$, we obtain a solution satisfying

$$\begin{aligned} f(S_e \cup T) &= f(B_j \cup T_j) \geq f(B_j) + (1 - e^{-\frac{\frac{1}{2}-S_j}{1-S_1-S_2}})f_{P_k}(L) - \frac{1}{k}f(P_k) \\ &\geq \text{OPT} \left(p_j + (1 - p_1 + p_2)(1 - e^{-\frac{\frac{1}{2}-S_j}{1-S_1-S_2}}) - \frac{p_1 + p_2}{k} \right). \end{aligned}$$

By Lemma 8, in one of these iterations we obtain a solution of value at least $\text{OPT}/3$, implying the statement of the theorem. \blacktriangleleft

Proof of Lemma 8. Let p_1, p_2, S_1, S_2 be values that satisfy the conditions in the lemma.

Denote $p = p_1 + p_2, d = p_1 - p_2$ and $r = e^{-\frac{\frac{1}{2}-S_1}{1-S_1-S_2}}$. It is easy to see that $e^{-\frac{\frac{1}{2}-S_2}{1-S_1-S_2}} = e^{-1}r^{-1}$. Define $V_1 = h(p_1, p_2, S_1, S_2)$ and $V_2 = h(p_2, p_1, S_2, S_1)$. By the definition of h and above definitions we get

$$V_1 = \frac{p+d}{2} + (1-p)(1-r) - \frac{p}{k}$$

and

$$V_2 = \frac{p-d}{2} + (1-p)(1 - e^{-1}r^{-1}) - \frac{p}{k}$$

Let $g_1(x) = \frac{p+d}{2} + (1-p)(1-x) - \frac{p}{k}$ and $g_2(x) = \frac{p-d}{2} + (1-p)(1 - e^{-1}x^{-1}) - \frac{p}{k}$. Clearly, $V_1 = g_1(r)$ and $V_2 = g_2(r)$. It is also easy to see that g_1 is decreasing and g_2 is increasing (for $x > 0$).

\triangleright **Claim 10.** It holds that $g_1(x^*) = g_2(x^*)$ where $x^* = \frac{d + \sqrt{d^2 + 4e^{-1}(1-p)^2}}{2(1-p)}$.

Proof. By rearranging terms we have $g_1(x) = g_2(x)$ if and only if $d = (1-p)(x - e^{-1}x^{-1})$, which holds for $x > 0$ if and only if $0 = (1-p)x^2 - dx - e^{-1}(1-p)$. The latter is a quadratic equation and $x^* > 0$ is a root. \triangleleft

If $r \geq x^*$, since g_2 is increasing, we have $V_2 = g_2(r) \geq g_2(x^*) = g_1(x^*)$, and if $r \leq x^*$, as g_1 is decreasing, we have $V_1 = g_1(r) \geq g_1(x^*)$. Therefore $\max(V_1, V_2) \geq g_1(x^*)$. By rearranging terms and substituting $k = 6$ we have

$$g_1(x^*) = 1 - \frac{p}{2} - \frac{\sqrt{d^2 + 4e^{-1}(1-p)^2}}{2} - \frac{p}{6} = 1 - \frac{2p}{3} - \frac{\sqrt{d^2 + 4e^{-1}(1-p)^2}}{2}$$

Our goal is to show that for $0 \leq p \leq \frac{2}{3}$, we have $1 - \frac{2p}{3} - \frac{1}{2} \cdot \sqrt{d^2 + 4e^{-1}(1-p)^2} \geq \frac{1}{3}$, or $\frac{2}{3}(1-p) - \frac{1}{2} \cdot \sqrt{d^2 + 4e^{-1}(1-p)^2} \geq 0$. By rearranging terms this is equivalent to showing

$$\frac{2}{3} \geq \sqrt{\frac{d^2}{4(1-p)^2} + e^{-1}}.$$

Consider two cases:

Case 1: $0 \leq p \leq \frac{1}{3}$. Since $|d| \leq p$ we have

$$\sqrt{\frac{d^2}{4(1-p)^2} + e^{-1}} \leq \sqrt{\frac{p^2}{4(1-p)^2} + e^{-1}} \leq \sqrt{\frac{1}{16} + e^{-1}} \leq \frac{2}{3}.$$

The last inequality follows since the function $\frac{x}{1-x}$ is increasing in the interval $[0, \frac{1}{3}]$.

69:12 Generalized Assignment via Submodular Optimization

Case 2: $\frac{1}{3} \leq p \leq \frac{2}{3}$, as $\frac{p+d}{2}, \frac{p-d}{2} \leq \frac{1}{3}$, we have that $|d| \leq \frac{2}{3} - p$. Therefore,

$$\sqrt{\frac{d^2}{4(1-p)^2} + e^{-1}} \leq \sqrt{\frac{(2/3-p)^2}{4(1-p)^2} + e^{-1}} \leq \sqrt{\frac{1}{16} + e^{-1}} \leq \frac{2}{3}.$$

The last inequality follows since the function $\frac{2/3-x}{1-x}$ is decreasing in the interval $[\frac{1}{3}, \frac{2}{3}]$.

In both cases we get $g_1(x^*) \geq \frac{1}{3}$, and as $\max(V_1, V_2) \geq g_1(x^*)$, the lemma follows. \blacktriangleleft

5 Filling Phase

In this section we prove Theorem 4. Define the size of bin j in assignment U as $s_j^U = \sum_{i \in U_j} s_i$. We first divide the bins and items into types. We say that a bin j is *full* if $s_j^U > 1$, *semi-full* if $\frac{1}{2} \leq s_j^U \leq 1$, and *semi-vacant* if $s_j^U < \frac{1}{2}$. An item $i \in I(U)$ is *big* if $s_i > \frac{1}{2}$; otherwise, i is *small*. Clearly, there are no big items in semi-vacant bins.

Informally, we use in the proof several types of resolution steps. Each step takes as input a full bin and possibly one or two semi-vacant bins, and reassigns some of the items into the bins while evicting others. These resolution steps ensure that the new assignment has at least half the profit of the original assignment, the assignment to any bin remains feasible, and only small items are evicted.

We apply the resolution steps repeatedly, but once a bin participated in a resolution step it may not participate in another one. We then prove that as long as there are full bins, one of the steps can be applied. Hence, by applying the resolution steps, we have a new assignment in which all bins are feasible, and the total profit is at least half the profit of the original assignment. To handle the evicted items, we note that as $s(I(U)) \leq m/2$ and all the evicted items are small, it is possible to assign the evicted items to bins without violating the capacity constraints.

Proof of Theorem 4. For any bin $1 \leq j \leq m$ and $A \subseteq I$, let $p_j(A) = \sum_{i \in A} p_{i,j}$ be the total profit gained from packing A into j . The first step is to resolve the violation of the capacity constraint in each full bin. We do that using four types of resolution steps. Each step takes a full bin and possibly one or two semi-vacant bins, modifies their contents and adds some small elements to a set V of *evicted* elements, that will be handled later. Throughout the discussion, we consider for a full bin j a partition of the elements in U_j into two feasible subsets, given by $\{A_j, B_j\}$. We use the following resolution steps.

1. Consider a full bin j such that U_j has no big elements. If $p_j(A_j) > p_j(B_j)$ then set $U'_j = A_j$ and evict B_j ($V := V \cup B_j$); otherwise, set $U'_j = B_j$ and evict A_j , ($V := V \cup A_j$). In both cases U'_j is feasible and $p_j(U'_j) \geq \frac{1}{2} \cdot p_j(U_j)$.
2. Now, suppose we have a full bin j such that U_j has a single big element, and a semi-vacant bin ℓ . Let $\{A^*, B^*\} = \{A_j, B_j\}$, such that the big element is in A^* . If $p_j(A^*) + p_\ell(U_\ell) > p_j(B^*)$, set $U'_j = A^*$, $U'_\ell = U_\ell$ and evict the elements in B^* ($V := V \cup B^*$). We note that in this case $p_j(U'_j) + p_\ell(U'_\ell) = p_j(A^*) + p_\ell(U_\ell)$. Otherwise, set $U'_j = B^*$ and $U'_\ell = A^*$, and evict all the elements in U_ℓ , ($V := V \cup U_\ell$). In this case we have $p_j(U'_j) + p_\ell(U'_\ell) \geq p_j(B^*)$. Therefore, in both cases have $p_j(U'_j) + p_\ell(U'_\ell) \geq \frac{1}{2} \cdot (p_j(U_j) + p_\ell(U_\ell))$.
3. Consider a full bin j such that U_j has two big elements, and a semi-vacant bin ℓ , such that one of the big elements has space in bin ℓ ; that is, there is a big element $i^* \in U_j$ such that $s_{i^*} + s_\ell^U \leq 1$.

Let $\{A^*, B^*\} = \{A_j, B_j\}$ such that $i^* \in A^*$. We note that there cannot be any other big element in A^* other than i^* .

If $p_j(B^*) + p_\ell(U_\ell) > p_j(A^*)$ set $U'_j = B^*$ and $U'_\ell = U_\ell \cup \{i^*\}$ (note that $s_\ell^{U'_\ell} \leq 1$). Also, evict all elements in $A^* \setminus \{i^*\}$. In this case we have $p_j(U'_j) + p_\ell(U'_\ell) \geq p_j(B^*) + p_\ell(U_\ell)$.

Otherwise, we set $U'_j = A^*$ and $U'_\ell = B^*$, and evict U_ℓ ($V := V \cup U_\ell$). In this case we have $p_j(U'_j) + p_\ell(U'_\ell) \geq p_j(A^*)$.

Thus, in both cases $p_j(U'_j) + p_\ell(U'_\ell) \geq \frac{1}{2} \cdot (p_j(U_j) + p_\ell(U_\ell))$.

4. Finally, consider a full bin j , such that U_j has two big elements, and two semi-vacant bins ℓ_1 and ℓ_2 . Recall A_j, B_j is a partition of the elements in U_j into two feasible subsets. If $p_j(A_j) + p_{\ell_1}(U_{\ell_1}) > p_j(B_j) + p_{\ell_2}(U_{\ell_2})$, set $U'_j = A_j$, $U'_{\ell_1} = U_{\ell_1}$ and $U'_{\ell_2} = B_j$, and evict U_{ℓ_2} . Thus, we have

$$p_j(U'_j) + p_{\ell_1}(U'_{\ell_1}) + p_{\ell_2}(U'_{\ell_2}) \geq p_j(A_j) + p_{\ell_1}(U_{\ell_1}).$$

Otherwise, set $U'_j = B_j$, $U'_{\ell_1} = A_j$ and $U'_{\ell_2} = U_{\ell_2}$ and evict U_{ℓ_1} . Here

$$p_j(U'_j) + p_{\ell_1}(U'_{\ell_1}) + p_{\ell_2}(U'_{\ell_2}) \geq p_j(B_j) + p_{\ell_2}(U_{\ell_2}).$$

And finally, we always have

$$p_j(U'_j) + p_{\ell_1}(U'_{\ell_1}) + p_{\ell_2}(U'_{\ell_2}) \geq \frac{1}{2} \cdot (p_j(U_j) + p_{\ell_1}(U_{\ell_1}) + p_{\ell_2}(U_{\ell_2})).$$

Each time we execute a step we mark the bins used in this step as *resolved*, and we do not consider them in the next steps. We first use Steps 1, 2, and 3, until none of them can be applied.

Consider the average size of the unresolved bins. When we start, there are m bins of average size no greater than half. Each of Steps 1, 2, and 3 reduces the size of the unresolved bins by at least one (as a full bin is removed) and reduces the number of bins by at most two. Therefore, the average size of the unresolved bins remains no more than half. Also, marking all the semi-full bins as resolved will preserve the property.

Let a be the number of unresolved full bins and c be the number of unresolved semi-vacant bins. Due to the average size of bins, we have $a \leq \frac{a+c}{2}$, therefore $a \leq c$. Hence, if we have a full bin, there must be a semi-vacant bin as well. As we used Steps 1, 2, and 3 to exhaustion, every full bin must have two big elements (no bin in U contains more than two big elements), and none of these big elements can fit into one of the semi-vacant bins.

Denote the minimal size of a semi-vacant unresolved bin by r . Then each of the full bins has two big elements of size greater than $1 - r$. Hence, we have $c \cdot r + 2a \cdot (1 - r) < \frac{a+c}{2}$, which leads to $2a(1 - r) - \frac{a}{2} < c(\frac{1}{2} - r)$, and

$$c > a \cdot \frac{2 - 2r - \frac{1}{2}}{\frac{1}{2} - r} = a \cdot \frac{3 - 4r}{1 - 2r} = a \cdot \left(\frac{2 - 4r}{1 - 2r} + \frac{1}{1 - 2r} \right) = a \cdot \left(2 + \frac{1}{1 - 2r} \right) > 2a,$$

implying that we can now run Step 4, until there are no more unresolved full bins.

We use the resolution steps to eliminate all the full bins. Every time we run such a step over a set of bins we lose at most half the profit of the bins participating in the step. As the total size of items in the assignment is bounded by $m/2$, we are guaranteed that it is possible to assign all the evicted elements to some bins. Thus, we are able to resolve the capacity overflow while losing at most half of the profit. ◀

6 Discussion and Future Work

In this paper we presented a $\frac{1}{6}$ -approximation algorithm for Group GAP, using a mild assumption on the size of each group. A key component in our result is an algorithm for submodular maximization subject to a knapsack constraint, which finds a solution occupying at most half the knapsack capacity, while the other half is *reserved* for later use. Our results leave several avenues for future work.

As mentioned above, Group GAP with no assumption on group sizes cannot be approximated within any constant factor. Yet, the maximum group size that still allows to obtain a constant ratio can be anywhere in $[\frac{m}{2}, \frac{2}{3}m]$. Thus, a natural question is: “Can our results be applied to instances with larger group sizes?” We note that the ratio stated in Theorem 5 may not hold already for instances in which group sizes can be at most $\frac{m}{2}(1 + \varepsilon)$, for some $\varepsilon > 0$. Indeed, for such instances, it may be the case that no set of groups of total size at most $m/2$ is “good” relative to the optimum. The existence of an algorithm that yields a constant ratio for such instances remains open.

While our result for submodular optimization with reserved capacity (Theorem 2) gives an optimal approximation ratio for the studied subclass of instances, we believe the result can be extended to other subclasses. In particular, we conjecture that for instances where each item has size at most $\delta > 0$, the approximation ratio approaches $1 - e^{-\frac{1}{2}}$ as $\delta \rightarrow 0$. Such result would immediately imply an improved approximation ratio for instances of Group GAP in which the total size of each group is bounded by δm . We defer this line of work to the full version of the paper.

Lastly, we introduced in the paper the novel approach of submodular optimization subject to a knapsack with reserved capacity constraint. We applied the approach along with a framework similar to the one developed in [1]. It would be interesting to investigate whether the approach can be used to improve the approximation ratio obtained in [1] for *all-or-nothing* GAP.

References

- 1 Ron Adany, Moran Feldman, Elad Haramaty, Rohit Khandekar, Baruch Schieber, Roy Schwartz, Hadas Shachnai, and Tami Tamir. All-Or-Nothing Generalized Assignment with Application to Scheduling Advertising Campaigns. *ACM Trans. Algorithms*, 12(3):38:1–38:25, 2016.
- 2 V. Balachandran. An Integer Generalized Transportation Model for Optimal Job Assignment in Computer Networks. *Operations Research*, 24(4):742–759, 1976.
- 3 Amotz Bar-Noy and George Rabanca. Tight approximation bounds for the seminar assignment problem. In *International Workshop on Approximation and Online Algorithms*, pages 170–182. Springer, 2016.
- 4 Marco Bender, Clemens Thielen, and Stephan Westphal. Packing items into several bins facilitates approximating the separable assignment problem. *Inf. Processing Letters*, 115(6-8):570–575, 2015.
- 5 Niv Buchbinder and Moran Feldman. Deterministic Algorithms for Submodular Maximization Problems. In *27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 392–403, 2016.
- 6 Niv Buchbinder and Moran Feldman. Submodular Functions Maximization Problems – A Survey. In Teofilo F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics (2nd Edition)*, volume 1, chapter 42. Chapman and Hall/CRC, 2018.
- 7 Grigori Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a Monotone Submodular Function Subject to a Matroid Constraint. *SIAM J. on Computing*, 40(6):1740–1766, 2011.

- 8 C. Chekuri and S. Khanna. A PTAS for the Multiple Knapsack Problem. *SIAM J. on Computing*, 35(3):713–728, 2006.
- 9 Lin Chen and Guochuan Zhang. Packing Groups of Items into Multiple Knapsacks. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 28:1–28:13, 2016.
- 10 Robert G. Cromley and Dean M. Hanink. Coupling land use allocation models with raster GIS. *Journal of Geographical Systems*, 1(2):137–153, 1999.
- 11 Uriel Feige. A Threshold of $\ln n$ for Approximating Set Cover. *J. ACM*, 45(4), July 1998.
- 12 Uriel Feige and Jan Vondrák. Approximation algorithms for allocation problems: Improving the factor of $1-1/e$. In *47th Annual IEEE Symposium on Foundations of Computer Science, FOCS'06*, pages 667–676, 2006.
- 13 M. Feldman, J. Naor, and R. Schwartz. A Unified Continuous Greedy Algorithm for Submodular Maximization. In *52nd Annual IEEE Symposium on Foundations of Computer Science, FOCS'11*, pages 570–579, 2011.
- 14 Lisa Fleischer, Michel X Goemans, Vahab S Mirrokni, and Maxim Sviridenko. Tight Approximation Algorithms for Maximum Separable Assignment Problems. *Math. Oper. Res.*, 36(3):416–431, 2011.
- 15 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 16 Ariel Kulik, Kanthi K. Sarpatwar, Baruch Schieber, and Hadas Shachnai. Generalized Assignment via Submodular Optimization with Reserved Capacity. *CoRR*, abs/1907.01745, 2019. [arXiv:1907.01745](https://arxiv.org/abs/1907.01745).
- 17 Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization*, volume 46. Cambridge University Press, 2011.
- 18 Hui Lin and Jeff Bilmes. Multi-document Summarization via Budgeted Maximization of Submodular Functions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, HLT'10*, 2010.
- 19 G. L. Nemhauser and L. A. Wolsey. Best Algorithms for Approximating the Maximum of a Submodular Set Function. *Math. Oper. Res.*, 3(3):177–188, 1978.
- 20 Kanthi K. Sarpatwar, Baruch Schieber, and Hadas Shachnai. Generalized Assignment of Time-Sensitive Item Groups. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 24:1–24:18, 2018.
- 21 David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62:461–474, 1993.
- 22 Maxim Sviridenko. A Note on Maximizing a Submodular Set Function Subject to Knapsack Constraint. *Operations Research Letters*, 32:41–43, 2004.
- 23 Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *40th Annual ACM Symposium on Theory of Computing, STOC, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 67–74, 2008.

Resilient Dictionaries for Randomly Unreliable Memory

Stefano Leucci 

Department of Algorithms and Complexity, Max Planck Institute for Informatics, Germany*

<https://www.stefanoleucci.com>

stefano.leucci@mpi-inf.mpg.de

Chih-Hung Liu 

Department of Computer Science, ETH Zürich, Switzerland

chih-hung.liu@inf.ethz.ch

Simon Meierhans

Department of Computer Science, ETH Zürich, Switzerland

mesimon@student.ethz.ch

Abstract

We study the problem of designing a dictionary data structure that is *resilient* to memory corruptions. Our error model is a variation of the faulty RAM model in which, except for constant amount of definitely *reliable* memory, each memory word is randomly *unreliable* with a probability $p < \frac{1}{2}$, and the locations of the unreliable words are unknown to the algorithm. An adversary observes the whole memory and can, at any time, arbitrarily *corrupt* (i.e., modify) the contents of one or more *unreliable* words.

Our dictionary has capacity n , stores $N < n$ keys in the optimal $O(N)$ amount of space, supports insertions and deletions in $O(\log n)$ amortized time, and allows to search for a key in $O(\log n)$ worst-case time. With a *global* probability of at least $1 - \frac{1}{n}$, all possible search operations are guaranteed to return the correct answer w.r.t. the set of uncorrupted keys.

The closest related results are the ones of Finocchi et al. [13] and Brodal et al. [6] on the faulty RAM model, in which all but $O(1)$ memory is unreliable. There, if an upper bound δ on the number of corruptions is known in advance, all dictionary operations can be implemented in $\Theta(\log n + \delta)$ amortized time, thus trading resiliency for speed as soon as $\delta = \omega(\log n)$.

Our construction does not need to know the value of δ in advance and remains *fast* and *effective* even when up to a constant fraction of the available memory is corrupted. Our techniques can be immediately extended to implement other data types (e.g., associative containers and priority queues), which can then be used as a building block in the design of other resilient algorithms. For example, we are able to solve the *resilient sorting* problem in our model using $O(n \log n)$ time.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases resilient dictionary, unreliable memory, faulty RAM

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.70

Funding Research supported by the SNSF (Swiss National Science Foundation) grant 200021_165524.

1 Introduction

Computing platforms sometimes exhibit temporary or permanent memory failures and they can be expected to become more frequent due to the challenge of providing high amounts of energy on an ever smaller scale, while simultaneously increasing the operation frequency. Most algorithms completely malfunction even if a single memory error occurs. For example, one may consider the *Mergesort* algorithm, where a constant fraction of elements can be placed out of order following a single corruption. Classical approaches to deal with these

* Part of this work was completed while the author was affiliated with ETH Zürich.



© Stefano Leucci, Chih-Hung Liu, and Simon Meierhans;

licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 70; pp. 70:1–70:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

faults involve data replication or the use of error-correcting code (ECC) memory, and typically require more computational resources (i.e., space and/or time) or dedicated hardware. This gave rise to a line of research focusing on the design of fast and compact algorithms and data structures that are *resilient* to memory faults, i.e., that provide provable guarantees on their output even when some memory words become corrupted.

A widely used model to capture this kind of corruption is the faulty RAM model [14], in which an algorithm has access to only a constant amount of *reliable* memory, while all the other words are *unreliable*. An adversary is then allowed to corrupt (i.e., change the value of) up to δ unreliable words,¹ and the algorithms' performances are evaluated as a function of δ . We consider a variation of the above model in which, in addition to the $O(1)$ reliable memory, each of the remaining words is unreliable with a certain probability $p < \frac{1}{2}$ but the algorithm is unaware of which of these locations are reliable. While both settings are theoretical abstractions of the more complex error patterns that can happen in the real-world, it is not hard to come up with examples that closely resemble our error model. For example, when a DRAM module is faulty, its contents are no longer reliable and read and write operations might produce corrupted values. To complicate things further, even though the faulty locations might be contiguous, processors map physical addresses to hardware locations using complex (and often undocumented) mapping functions.² This increases memory access parallelism but has the side effect of distributing the unreliable locations over the whole address space.

We focus on the problem of designing a resilient data structure implementing the *dictionary* abstract data type, that is, we want to maintain a set of up to n keys under insertion and deletions so that we can answer membership queries. While it is easy to adapt any classical (non-resilient) data structure to our model with a multiplicative blow up of $\Theta(\log n)$ in the time and space complexities, one might wonder whether provably good guarantees w.r.t. the number of corruptions can be achieved using the asymptotically optimal $O(n)$ amount of space. In the rest of this paper we show that the answer is indeed affirmative.

Our model

Our model of computation is a random access machine [1] in which each word of memory is either *reliable* or *unreliable*. The memory itself is split into two regions:

- A *safe* region of $O(1)$ words (representing, e.g., processor registers) that are always *reliable*.
- A region containing all of the remaining words (representing, e.g., the main memory). Each of these words is independently *reliable* with probability $1 - p$ and *unreliable* with probability p , for some constant $p < \frac{1}{2}$.

Unreliable words can be affected by *corruption* phenomena, i.e., their contents can unexpectedly change. Except for the $O(1)$ words in the safe memory region, an algorithm is unaware of which memory locations are unreliable and is unable to detect whether the value stored in an unreliable location has been corrupted. Following [14], we adopt a worst case approach to corruptions: a computationally-unbounded adversary knows the algorithm that is being executed, can observe the contents of the whole memory, and can distinguish reliable from unreliable words. The adversary can, at any point during the execution of the algorithm, *simultaneously* corrupt the values stored in one or more *unreliable* words.

¹ This implicitly adopts a worst-case approach, so that all positive results also hold in the easier case of random, non-malicious corruptions.

² Finding techniques to reliably discover the mapping functions is an area of active research. Proposed methods use thermal and timing data to correlate physical addresses to memory locations [19, 18, 24].

Our results and techniques

We design a *resilient dictionary* that stores up to n keys using the optimal amount of space, supports insertions, deletions, and membership queries (in the following **insert**, **delete**, and **search**) in $O(\log n)$ amortized time and guarantees *with high probability* (w.h.p.) that, regardless of the number δ of corruptions, all **search** operations work reliably on all uncorrupted keys. More precisely, the **search** operation behaves as follows: if the searched key k belongs to the dictionary and is uncorrupted, then the **search** operation correctly returns **yes**. If k does not belong to the dictionary and no corrupted key equals k , the returned answer is guaranteed to be **no**. In the remaining cases the answer might be either **yes** or **no**.

As a comparison, the closest related results are the ones of Brodal et al. [6] and Finocchi et al. [13] on the faulty RAM model. There the authors show how, given δ , it is possible to build a dictionary that is resilient to up to δ word corruptions, uses $O(n)$ space, and whose **insert**, **delete**, and **search** operations require $O(\log n + \delta)$ amortized time, which is tight.³

Crucially, the task of selecting a good upper bound δ on the number of corruptions at design time can turn out to be problematic: if δ is too small then the resulting dictionary can abruptly fail when the $(\delta + 1)$ -th corruption occurs while, if $\delta = \omega(\log n)$, one is forced to trade-off resiliency for speed as the additive term of δ now becomes dominant in the time complexity.

Notice that, in our setting, the *expected* number of unreliable locations is $pn = \Theta(n)$, implying that the *actual* number of unreliable locations is $\Theta(n)$, except for an exponentially vanishing probability. Since δ can be as large $\Theta(n)$ (think, e.g., of our faulty DRAM-example), a direct instantiation of the results of [6] and [13] yields the same bounds of the trivial dictionary that uses $O(n)$ space and time per operation (just store all the keys in an unsorted array). Nevertheless, one might wish for a dictionary that remains resilient w.r.t. a constant fraction of key corruptions and whose operations do not bear the exorbitant cost of $O(n)$ time per operation. Our results shows that this is indeed possible: when the unreliable locations are random fraction of the available memory, it is possible handle *any (unknown) number* δ of corruptions with the same asymptotic guarantees obtained by the existing fault-tolerant dictionaries in the faulty RAM model when $\delta = O(\log n)$.

From a high-level point of view, [13] *buffers* the dictionary keys into groups of size $\Theta(\delta)$, and organizes the resulting groups into a pointer-based AVL tree. A core subroutine then consists of **locating** the group responsible for a given key in $O(\log n + \delta)$ time. Since the corruption of any auxiliary information associated with a group might lead the search astray, the corresponding variables are replicated $\Theta(\delta)$ times. A first difficulty to overcome in our model is that of removing the dependency from δ . We do so by using a different locate procedure that is based on a suitable walk over the tree vertices. The behavior of the walk is guided by the observed memory contents, yet we prove that there is an absolute probability of at least $1 - \frac{1}{n}$ that no possible walk can be misled by the adversary, even when all the unreliable memory locations are corrupted. Another difficulty is intrinsic in pointer-based structures: if the address stored in a pointer cannot be reliably recovered, the whole pointed object becomes inaccessible. The natural way to deal with this problem involves reading multiple copies of the pointer. However, since reading only $o(\log n)$ copies would still allow the adversary to corrupt the pointer with a probability larger than $\frac{1}{n}$, this would lead to a

³ The dictionary of [13] is randomized and the above bounds hold in expectation, while the one of [6] is deterministic. The dictionary of [13] can be made deterministic by slightly worsening the δ additive term in the amortized complexity to $\delta^{1+\epsilon}$, for a constant $\epsilon > 0$ of choice.

slowdown of $\Omega(\log n)$. We avoid this problem altogether by using a *dynamic* binary search tree (BST) that is *nearly balanced*, i.e., whose height is at most an *additive constant* larger than the optimal one [7]. This tree is then embedded into a *static* and *pointer-free* complete BST. We are now left with one final technical obstacle: since rebalancing such a dynamic tree following an update operation is more expensive w.r.t. its AVL counterpart, we need to employ a more elaborate 2-level buffering scheme. Namely, we group keys into $O(\frac{n}{\log n})$ *pages* of capacity $O(\log n)$ which are in turn arranged into sorted *folders* consisting of $O(\log n)$ pages (i.e., $O(\log^2 n)$ keys) each.

We remark that the high probability bound on the correctness of our dictionary does not depend on the number of operations performed. In other words, there is a *global* probability of at least $1 - \frac{1}{n}$ that the operations in any (arbitrarily long) sequence of **inserts**, **deletes**, and **searches**, are *all* jointly correct. The following Theorem summarizes our result:

► **Theorem 1.** *A resilient dictionary with capacity n can be constructed in $O(1)$ worst-case time. Search, and update operations (i.e., **insert** and **delete**) require $O(\log n)$ worst-case and $O(\log n)$ amortized time, respectively. The space required is $O(N)$, where N is the number of elements currently stored in the dictionary. All operations performed during the lifetime of the dictionary are (jointly) correct with probability at least $1 - \frac{1}{n}$.*

Although our focus is on the dictionary abstract data type, our techniques immediately extended to other data types (e.g., associative containers and priority queues), as we discuss in Section 6.

Other related results

Apart from the already mentioned results of [13] and [6], several other algorithms and models to deal with faulty computations have been proposed, with considerable attention paid to the problems of sorting and searching. In the faulty RAM model, the *resilient sorting* problem asks to output a permutation of an input set of n elements such that the set of uncorrupted elements forms a sorted subsequence, while the *resilient searching* problem asks to locate an uncorrupted key k in such a sorted sequence, if it exists. Roughly speaking, when δ corruptions happen, one can resiliently sort and search at the cost of an additional *additive* term in the time complexity that depends only on δ . More specifically, one can search in $\Theta(\log n + \delta)$ time [6, 14] and sort in $O(n \log n + \delta^2)$ time which can be improved to $O(n + \delta^2)$ when elements are polynomially-bounded integers [12]. Moreover, for $\delta = \omega(\sqrt{n \log n})$, all resilient sorting algorithms must require $\omega(n \log n)$ time in the general case [14].

Another popular model dealing with faults considers the elements as opaque objects possessing an unknown linear order that needs to be discovered or approximated through *pairwise noisy comparisons*, i.e., comparisons that can sometimes be incorrect. If comparison errors happen randomly and resampling is allowed (i.e., a comparison can be repeated multiple times and the results are independent) then one can sort and search, w.h.p., in the optimal $O(n \log n)$ and $O(\log n)$ time, respectively [11]. If resampling is not allowed then one cannot reliably reconstruct the correct linear order. Nevertheless, it is possible to compute a permutation in which each element is misplaced by at most $O(\log n)$ positions, which is asymptotically optimal. Several solutions have been designed for this problem [5, 25, 20, 15, 16, 17] culminating in an optimal $O(n \log n)$ time algorithm. If errors are adversarial then $\Omega(n \log n + \delta n)$ comparisons are needed, and this is tight [3, 21, 22]. Adversarial errors in searching are studied in the context of Rényi-Ulam Games, in which a questioner needs to guess an element from a domain by asking comparison questions to a lying responder. An extensive collection of results exists, depending on the considered domain and on the constraints on the responder's lies. We refer the interested reader to [23] and [9] for a survey and a monograph.

In [2] the authors define the *fault-tolerance* of a data structure as the maximum ratio between the amount of data lost as a consequence of a number δ of corruptions and δ . They provide implementations of linked lists, stacks, and binary search trees with a fault-tolerance of $O(\log \delta)$ or $O(1)$, depending on the specific construction used, while losing only constant *multiplicative* factors in the time and space requirements. We remark that in [2], unlike both our dictionary and the ones in [13, 6], uncorrupted keys can also be lost until the whole data structure is reconstructed. If $\Theta(\delta)$ words of safe memory are available then, instead of the multiplicative overhead of [2], one can pay only an *additive* overhead of $\tilde{O}(\delta)$ in the time and space complexities, and the data structure can be reconstructed in linear time [8]. Similarly to [13] and [6], and differently from our dictionary, both [2] and [8] require the value of δ to be known in advance.

2 Preliminaries

For simplicity we assume that the capacity n of our dictionary is a sufficiently large power of 2, so that $\log n$ is a sufficiently large integer and we do not have to deal with rounding. We also assume, w.l.o.g., that $p \leq \frac{1}{1024}$.⁴ A concept of *resilient variable* similar to the one used in [13] will also be useful in the sequel. For our purposes, a resilient variable x consists of $20 \log n + 1$ consecutive copies x_1, x_2, \dots of a classical variable. A resilient variable can be written (i.e., assigned to) in $O(\log n)$ time and constant space by simply writing the intended value to each x_i . We can then read the value stored in x either resiliently or non-resiliently: a non-resilient read amounts to returning the (possibly corrupted) value of a single x_i ; a resilient read returns the majority value among those stored in all the copies $x_1, \dots, x_{20 \log n + 1}$, or fails if no majority value exists. Notice that, if at least $10 \log n + 1$ copies are uncorrupted, the majority value coincides with the one previously stored in x and can be computed in $O(\log n)$ time using a constant number of words in safe memory (see, e.g., the Boyer-Moore majority vote algorithm [4]). An easy Chernoff-bound argument shows that this is indeed the case for *all possible read operations that can be performed*, w.h.p, as the following Lemma states (a formal proof will appear in the full version of the paper).

► **Lemma 2.** *Suppose that the number of words used to store replicated variables is $O(n)$. With probability at least $1 - n^{-2}$, all resilient read operations return the previously stored value.*

In the following we assume that all resilient read operations succeed as we will eventually union bound the success probability of the other operations in our dictionary with that of Lemma 2. Finally, we use $-\infty$ (resp. $+\infty$) to denote a key smaller than (resp. the largest among) all valid keys that can be inserted in our dictionary, and \perp as a placeholder for some invalid key value.

The paper is organized in a bottom-up fashion: Section 3 describes *pages*, which are groups of $O(\log n)$ keys; In Section 4 we further group pages into sorted *folders* and show how to *quickly* locate pages, while Section 5 uses pages and folders to build our dictionary. In Section 6 we summarize our results and provide some concluding remarks.

⁴ For any constant $\epsilon > 0$ and $p \in (\frac{1}{1024}, \frac{1}{2} - \epsilon]$, we can simulate the required error probability by storing $\lceil 30\epsilon^{-2} \rceil$ copies of each word, with a strategy similar to the one described in the rest of this section.

3 Pages

Each page v is associated with a contiguous interval $I(v) = (L_v, R_v]$ of keys and will be responsible for storing a set $K(v)$ of $O(\log n)$ keys, where all *uncorrupted* keys in $K(v)$ will belong to $I(v)$. A page π is implemented as contiguous array $A(\pi)$ of capacity $6 \log n$ and three replicated variables storing: the endpoints L_π and R_π of $I(\pi)$, and the *size* $|K(\pi)|$ of π . The keys in $K(\pi)$ are stored in the first $|K(\pi)|$ positions of $A(\pi)$ in an arbitrary order. If $|K(\pi)| = 0$ (resp. $|K(\pi)| = 6 \log n$) we say that π is empty (resp. full).

Pages will support five basic operations, namely **search**, **insert**, **delete**, **split**, and **merge**, as detailed in the following.

Search. Given $k \in I(v)$, the **search** operation determines whether $k \in K(\pi)$. We simply resiliently read the variable $|K(v)|$ and compare k with the first $|K(v)|$ elements of $A(\pi)$. If any (resp. no) element compares equal to k we report that k belongs to (resp. does not belong to) π . Clearly, this requires $O(\log n)$ worst-case time.

Insert. The **insert** operation adds a key $k \in I(\pi)$ to $K(\pi)$ and is only legal if π is not full. We first **search** for k and, if $k \in K(\pi)$, we are immediately done. Otherwise we write k in the $|K(\pi)|$ -th position of the array storing the set $K(v)$, where $|K(\pi)|$ is read resiliently. The time required is $O(\log n)$ in the worst-case.

Delete. The **delete** operation removes a key k from $K(\pi)$. If $k \notin K(\pi)$, then π is unaltered. The operation requires $O(\log n)$ time: after a resilient read of $|K(v)|$, we perform linear search on $A(\pi)$ and, if k is found in some position i , we swap the i -th and the $|K(v)|$ -th element of $A(\pi)$ and decrement $|K(v)|$ by 1 (which requires a replicated write operation). Due to corruptions, $A(\pi)$ might contain multiple occurrences of k , in which case exactly one of them is removed.

Split. The **split** operation destroys π and returns (the addresses of) two newly created pages π_1, π_2 such that: (i) $I(\pi_1) \cap I(\pi_2)$ is the empty interval; (ii) $I(\pi_1) \cup I(\pi_2) = I(\pi)$; (iii) $|K(\pi_1)| = \lfloor |K(\pi)|/2 \rfloor$ and $|K(\pi_2)| = \lceil |K(\pi)|/2 \rceil$; (iv) $K(\pi_1) \cup K(\pi_2) = K(\pi)$.

The difficulty of the **split** operation lies in computing a key $x \in [L_\pi, R_\pi]$ that allows to split $I(v)$ into two sub-intervals $I(\pi_1) = (L_\pi, x]$ and $I(\pi_2) = (x, R_\pi]$ while preserving the bounds on the sizes of π_1 and π_2 . We create a new page π_1 and we populate $A(\pi_1)$ by repeating the following iterative procedure $\lfloor |K(\pi)|/2 \rfloor$ times: in iteration i we search for the smallest key $k_i \in K(v)$, we remove k_i from $K(\pi)$ (in a way similar to our implementation of the **delete** operation), and we add it into the i -th position of $A(\pi_1)$. Finally, we rename π as π_2 , we choose $x = \min\{R_\pi, \max\{L_\pi, k_1, k_2, \dots, \}\}$, and we set $I(\pi_1)$ and $I(\pi_2)$ accordingly. It is easy to see that the split operation requires $O(\log^2 n)$ time in the worst case. The following lemma, whose proof is deferred to the full version of the paper, shows the uncorrupted keys are correctly partitioned among π_1 and π_2 .

► **Lemma 3.** *Let k be an uncorrupted key belonging to π before the **split** operation. If $k \leq x$ (resp. $k > x$) then k belongs to $K(\pi_1)$ (resp. $K(\pi_2)$) after the **split** operation.*

Merge. The **merge** operation can be thought of as the opposite of the **split** operation. It requires two pages π_1 and π_2 as inputs such that (i) $I(\pi_1) \cap I(\pi_2)$ is the empty interval, (ii) $I(\pi_1) \cup I(\pi_2)$ is a contiguous interval, and (iii) $|K(\pi_1)| + |K(\pi_2)| \leq 6 \log n$; and produces the following effects: it adds all keys in $K(\pi_2)$ to $K(\pi_1)$, updates $I(\pi_1)$ to $I(\pi_1) \cup I(\pi_2)$, and destroys π_2 . Notice that the order of π_1 and π_2 determines which of the two pages is destroyed.

Pages π_1 and π_2 are merged as follows: first we resiliently read $|K(\pi_1)|$ and $|K(\pi_2)|$, and append the $|K(\pi_2)|$ keys in π_2 in positions $|K(\pi_1)| + 1, \dots, |K(\pi_1)| + |K(\pi_2)|$ of $A(\pi_1)$. We then update $|K(\pi_1)|$ to $|K(\pi_1)| + |K(\pi_2)|$ and $I(\pi_1)$ to $(\min\{L_{\pi_1}, L_{\pi_2}\}, \max\{R_{\pi_1}, R_{\pi_2}\})$ where $L_{\pi_1}, L_{\pi_2}, R_{\pi_1}, R_{\pi_2}$ are also read resiliently. One can easily check that the overall time complexity of a merge operation is $O(\log n)$ in the worst case.

4 Folders

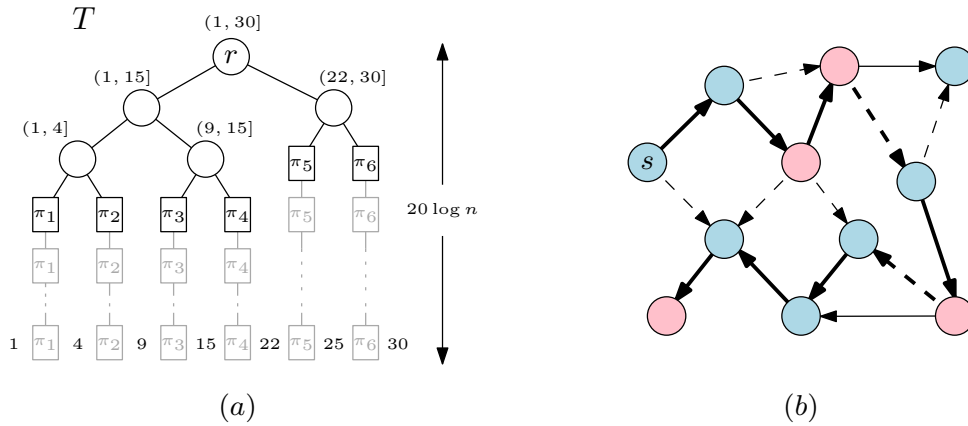
Folders are *sorted* arrays of capacity $6 \log n$ in which elements are *pages* having pairwise disjoint intervals. Each folder F additionally stores two replicated variables holding the number $|F|$ of pages currently in the array, and the overall number $|K(F)|$ of keys stored in (the pages of) F . With a slight abuse of notation we say that $\pi \in F$ if π is one of the pages of F . We denote by $K(F)$ the union of all sets $K(\pi)$ for $\pi \in F$. A folder F will always contain at least one page and, similarly to pages, it is (implicitly) associated with an interval $I(F) = (L_F, R_F]$. When F is first created it consists of a single *innate* empty page π with $I(\pi) = I(F)$. In general, each $\pi \in F$ is responsible for a certain sub-interval of $I(F)$ (formally, the intervals $I(\pi)$ are pairwise disjoint and satisfy $\cup_{\pi \in F} I(\pi) = I(F)$). Moreover, all pages except the first one will always contain at least $\log n + 1$ keys, and hence we immediately have that $|K(F)| \geq (|F| - 1) \cdot (\log n + 1)$ and that a folder can always accommodate at least $(6 \log n - 1)(\log n + 1) > 6 \log^2 n$ keys. In fact, we guarantee that F will always contain at most $6 \log^2 n$ keys. Quite naturally, we say that a folder F is *empty* if it contains no keys, and *full* if it contains exactly $6 \log^2 n$ keys. If F is not empty, then its first page will also be non-empty.

The pages in the array of F are sorted w.r.t. the order relation \prec implicitly defined by the order of their intervals, i.e., we say that two pages π_1, π_2 are such that $\pi_1 \prec \pi_2$ iff $I(\pi_1)$ precedes $I(\pi_2)$ ($\pi_1 \preceq \pi_2$, $\pi_1 \succ \pi_2$, and $\pi_1 \succeq \pi_2$ are defined accordingly). As a consequence, there is no need to explicitly store $I(F)$ since its left and right endpoints coincide with those of the first and last page in F , respectively.

We now discuss the operations supported by folders, most of which are similar to the ones for pages, on which they rely.

Locate and Search. Given a key $k \in I(F)$, the **locate** operation returns the unique page $\pi \in F$ such that $k \in I(\pi)$. Once the **locate** operation is available, one can easily **search** for a key $k \in I(F)$ by first **locating** the page π such that $k \in I(\pi)$ and then **searching** for k in π .

Notice that the **locate** operation can be easily implemented in $O(\log^2 n)$ worst case time, w.h.p., by binary searching for the page π while reading all the replicated variables resiliently. We now show that it is possible to reduce the time complexity to $O(\log n)$ by using a technique similar to the one of [11], where an element in a sorted array A is located through a random walk over the contiguous sub-intervals induced by the elements in A . Such a random walk will observe inconsistent results *independently at random* and is allowed to backtrack when this happens. Moreover, to guarantee a *high probability* of success, $\Omega(\log |A|)$ consistent observations are required before an element is returned. However, the analysis of [11] is not immediately suitable for our case due to two main obstacles, namely: (i) we want the high probability bound on the success probability to hold *jointly* for all **locate** operations performed during the lifetime of our dictionary, and (ii) since the adversary has complete control over the unreliable memory locations and a complete knowledge of the memory state, he can cause the inconsistent results to become correlated (e.g., by steering



■ **Figure 1** (a) An example tree T used by our random walk to implement the `locate` operation. (b) An example graph G having maximum out-degree $\Delta = 4$. Preferential edges are drawn with solid lines. A traversable path of type 2 and length 9 is highlighted in bold.

the walk towards corrupted memory locations). We overcome these obstacles by relating all the problematic realizations of all possible random walks to a collection of paths in a suitable graph. We then show that, with high probability, no possible set of memory corruptions can cause any such path to become viable.

The locate algorithm. We start by considering an almost-complete binary tree⁵ T in which the i -th leaf π_i corresponds to the i -th page of F , and an internal node v represents the (contiguous) interval $I(v) = (L_v, R_v]$ obtained by the union of all intervals $I(\pi)$ where π is a leaf of the subtree T_v of T rooted at v . Then, we augment such a tree by appending, to each π_i , a path P_{π_i} consisting of $20 \log n - d(\pi_i)$ copies of π_i itself, where $d(\pi_i) \in \{\lfloor \log 6n \rfloor, \lceil \log 6n \rceil\}$ is the depth of vertex π_i in T . See Figure 1 (a) for an example.

It is important to remark that T does not need to be explicitly constructed since each vertex $v \in V(T)$ can be represented by a triple of integers (i, j, h) , where i (resp. j) is the index, in the array of pages, of the smallest (resp. largest) leaf (i.e., page) in T_v , and h is the depth of v in T .

We now perform a discrete-time *walk* on T as follows: initially the current vertex v of the walk coincides with the root r of T and, at the generic i -th step, we *walk* from v to one of its neighbors. More precisely, if $I(v)$ is obtained by the union of all the intervals $I(\pi)$ for $\pi^- \preceq \pi \preceq \pi^+$ we read the i -th copy ℓ (resp. r) of the replicated variable L_{π^-} (resp. R_{π^+}), we check whether $\ell < k \leq r$ and, if that is *not* the case, we *walk* from v to the parent of v in T (in the special case $v = r$ we “walk” from v to itself). Otherwise, if v has only one child u in T , we walk from v to u . Finally, if v has two children u_1 and u_2 in T where $I(u_1)$ precedes $I(u_2)$ and is obtained by the union of all the intervals $I(\pi)$ for $\pi_1^- \preceq \pi \preceq \pi_1^+$, we compare k with the i -th copy x of $R_{\pi_1^+}$. If $k \leq x$ we walk to u_1 , otherwise we walk to u_2 .

We continue the walk for $20 \log n$ steps so that we are guaranteed that all v , except possibly for the one reached after the last step, have at least one child. If the vertex v reached at the end of the walk belongs to a path P_π , we return π . We say that the `locate` operation *fails* if either the v does not belong to any path P_π , or if $k \notin I(\pi)$.

⁵ A binary tree of height h is almost-complete if it is complete up to the $(h - 1)$ -th level. Moreover, we also assume that all the leaves on the h -th level are left-justified.

Analysis. Let us consider the following related problem. We are given a directed acyclic graph G having maximum out-degree at most $\Delta \geq 2$, a vertex $s \in V(G)$, and a probability $\rho \leq \frac{1}{32(\Delta-1)}$. Moreover, each non-sink vertex $v \in V(G)$ has exactly one *preferential* outgoing edge $(v, v^*) \in E(G)$ and we define E^* as the set of all preferential edges. The vertices of G are independently colored either red or blue with probability at most ρ and at least $1 - \rho$, respectively. Finally, we say that a path $P = \langle v_0, v_1, \dots, v_k \rangle$ in G is *traversable* if, for every $v_i \in P$ with $i = 0, \dots, k-1$, we have that v_i is red, or (v_i, v_{i+1}) is preferential, or both. See Figure 1 (b) for an example. We are now ready to state the following:

► **Lemma 4.** *Let ℓ be a multiple of 4. The probability that G contains a traversable path P of length ℓ from s such that $|E(P) \cap E^*| \geq \frac{\ell}{4}$ is at most $2^{-\frac{\ell}{4}+1}$.*

Proof. We will count the number of possible paths of length ℓ from s by grouping them according to the number of non-preferential edges they use. More precisely, we say that the *type* of a path $P = \langle s = v_0, v_1, \dots, v_\ell \rangle$ in G is κ if $|\{v_i^* \neq v_{i+1} : i \in \{0, \dots, \ell-1\}\}| = \kappa$, where (v_i, v_i^*) is the unique preferential edge outgoing from v_i . Notice that each path of type κ can be described by a set of κ pairs: $\{(\tau_1, a_1), \dots, (\tau_\kappa, a_\kappa)\}$ where (τ_i, a_i) signifies that when the path reaches v_{τ_i} , it continues towards the a_i -th non-preferential outgoing edge from v_{τ_i} (according to some fixed arbitrary order of the edges). This immediately implies that there are at most $\binom{\ell}{\kappa} (\Delta-1)^\kappa$ paths of type κ . Moreover, for any such path to be traversable, it must happen that all the vertices v_i with $i \in \{\tau_1, \dots, \tau_\kappa\}$ are red (while the remaining vertices can be either blue or red). This happens with probability at most ρ^κ , and hence the probability that there exists at least one traversable path of type κ is at most $\binom{\ell}{\kappa} (\Delta-1)^\kappa \rho^\kappa$. By summing over all $\kappa \in [\ell/4, \ell]$ we obtain:

$$\sum_{\kappa=\ell/4}^{\ell} \binom{\ell}{\kappa} (\Delta-1)^\kappa \rho^\kappa \leq \left(\sum_{\kappa=\ell/4}^{\ell} \binom{\ell}{\kappa} \right) \cdot \left(\sum_{\kappa=\ell/4}^{\infty} (\Delta-1)^\kappa \rho^\kappa \right) \leq \frac{2^\ell (\Delta-1)^{\frac{\ell}{4}} \rho^{\frac{\ell}{4}}}{1 - (\Delta-1)\rho} < 2^{-\frac{\ell}{4}+1}. \quad \blacktriangleleft$$

► **Lemma 5.** *With probability at least $1 - n^{-3}$ no `locate` operation on folders fails.*

Proof. We relate our random walk on T to the traversable paths in a suitable graph G . Let t be the unique leaf of T such that the key k to locate belongs to the corresponding page π and direct each edge of T towards t . We define G as the graph whose vertex set consists of $\ell + 1$ copies $u^{(1)}, u^{(2)}, \dots$ of each vertex $u \in V(T)$ and whose edge set contains, for all $i = 1, \dots, \ell$, the edge $(r^{(i)}, r^{(i+1)})$ and all the edges $(u^{(i)}, v^{(i+1)})$ where (u, v) is a directed edge in T . We color a vertex $u^{(i)}$ of G red if the i -th copy of at least one of the (at most 3) replicated variables accessed when a step of the random walk is performed from vertex $u^{(i)}$ is in an unreliable memory location (notice that these replicated variables correspond to the page associated with u and, possibly, with a child of u in T). For each non-sink vertex $u^{(i)}$ of G we let (u, v) be the unique (directed) edge outgoing from u in T and we choose the edge from $u^{(i)}$ to $v^{(i+1)}$ as the preferential edge from $u^{(i)}$ in G .

The graph G has maximum out-degree 3 and hence, once we choose $\Delta = 3$, $s = r$, $\rho = \frac{1}{128} > 3p$, and $\ell = 20 \log n$, we can invoke Lemma 4 to conclude that all the traversable paths from s in G of length $20 \log n$ use less than $5 \log n$ non-preferential edges with probability at least $1 - 2n^{-5}$. Since any sequence of vertices $\langle r = v_1, v_2, \dots, v_{\ell+1} \rangle$ corresponding to a realization of our random walk induces a traversable path $P = \langle s = v_1^{(1)}, v_2^{(2)}, \dots, v_{\ell+1}^{(\ell+1)} \rangle$ in G we know that, with the aforementioned probability, all possible random walks on T will use at least $15 \log n$ edges directed towards t . Each such edge decreases the distance in T from the current vertex of the walk to t by at least 1, while any other edge increases such a distance by at most 1. Since the distance $d_T(r, t)$ from r to t in T is $20 \log n$ by

70:10 Resilient Dictionaries for Randomly Unreliable Memory

construction, we conclude that the final vertex v of the walk must satisfy $d_T(v, t) \leq 10 \log n$ or, in other words, v lies on P_π and the `locate` operation is successful. The claim follows by noticing that, once the locations of the unreliable words are fixed, at most $O(n \log^2 n)$ distinct (colored) graphs G can exist, namely those corresponding to the $O(\log n)$ possible different sizes of T , to the $O(\log n)$ different positions of the sought leaf t in T , and to the $O(n)$ memory locations at which a folder can be stored. ◀

Insert. The `insert` operation adds a key $k \in I(F)$ into $K(F)$, and it is only legal when F is not full. We first `locate` the page π responsible for k and we say that the insert operation *targets* π . Then, if π is not full, we simply `insert` k into π , we update $|K(F)|$, and we are done. Otherwise, we say that the insert operation is *expensive*. Let i be the position of π in the array of pages. We move all pages in positions $i + 1, \dots, |F|$ one position to the right (so that the page originally in position j is now in position $j + 1$), and increment $|F|$ by 1.⁶ Every read and write operation on the replicated variables of the moved pages is performed resiliently, i.e., a replicated variable is moved by first resiliently reading its value x from the old memory location and then resiliently writing x to the new memory location. We now `split` π into two pages π_1 and π_2 containing $3 \log n$ keys each, and we store them in positions i and $i + 1$, respectively. Finally, we `insert` k into the unique page $\pi' \in \{\pi_1, \pi_2\}$ such that $k \in I(\pi')$, and we update $|K(F)|$.

Overall, an `insert` operation requires $O(\log n)$ worst-case time if it is not expensive and $O(\log^2 n)$ worst-case time otherwise.

Delete. The `delete` operation removes a key k from $K(T)$ if $k \in K(T)$. If $k \notin K(T)$ the operation does nothing. We first `locate` the page π responsible for k and we say that the delete operation *targets* π . We now try to `delete` k from π and, if k was not found in $K(\pi)$ we are already done. If k is found, then we decrement $|K(F)|$ by 1 and proceed differently depending on the size and on the position of π in the pages array of F . Whenever, after the deletion, any of the following conditions is true, no further work is necessary: (D1) $|K(\pi)| > \log n$; or (D2) π is the only page of F ; or (D3) π is the first page of F and is non-empty. Otherwise, we say that the delete operation is *expensive* and we distinguish two cases. If (i) π is the first page of F , (ii) F has at least 2 pages, and (iii) π is empty, we let π' be the second page in F and we delete π from F as follows: We set the left endpoint of $I(\pi')$ to the left endpoint of $I(\pi)$ (effectively updating $I(\pi')$ to $I(\pi) \cup I(\pi')$), and we shift every page of F other than π one position to its left, thus overwriting and destroying π . Finally, we decrement the value of $|F|$ by 1.

The complementary case is the one in which $|K(\pi)| \leq \log n$ and π is not the first page of F , which implies that $|K(\pi)| = \log n$. We let π' be the page preceding π in F and we further distinguish two sub-cases depending on the value of $|K(\pi')|$:

- If $|K(\pi')| \leq 4 \log n$, we `merge` π' and π . Since this operation destroys π , we shift all pages $\pi'' \in F$ such that $\pi'' \succ \pi$ by one position to their left, and we decrement $|F|$ by 1. Notice that π' now contains at most $4 \log n + \log n = 5 \log n$ keys.
- If $|K(\pi')| > 4 \log n$ we first `split` π' (destroying it) into π_1 and π_2 . We then `merge` π_2 and π (destroying π). We store π_1 in place of π' and π_2 in place of π . Notice that π_1 now contains at least $2 \log n$ and at most $3 \log n$ keys, while π_2 contains at least $2 \log n + \log n = 3 \log n$ and at most $3 \log n + \log n = 4 \log n$ keys.

⁶ Notice that, before the `insert` operation, we necessarily had $|F| < 6 \log n$ as otherwise $|K(F)|$ is at least $(6 \log n - 1)(\log n + 1)$, which violates our invariant on the number of keys stored in F .

Similarly to the **insert** operation, when pages are moved in the array of F , all their resilient variables are read and written resiliently. Overall, the required worst-case time is $O(\log n)$ if the operation is not expensive, and $O(\log^2 n)$ otherwise.

Split. If $|K(F)| \geq 12 \log n$, the **split** operation destroys F and returns two new folders F_1 and F_2 that together contain the same set of keys of F and such that: $I(F) = I(F_1) \cup I(F_2)$, $I(F_1) \cap I(F_2) = \emptyset$, and $|K(F)|/2 \leq |K(F_1)| < |K(F)|/2 + 6 \log n$ (implying $|K(F)|/2 - 6 \log n < |K(F_2)| \leq |K(F)|/2$), and all the uncorrupted keys $k \in K(\pi)$ belong to the unique folder $F' \in \{F_1, F_2\}$ such that $k \in I(F')$.

To implement the operation, we look for the first page π such that $\sum_{\pi' \preceq \pi} |K(\pi')| \geq |K(F)|/2$. We then construct the folders F_1 and F_2 , where T_1 contains all the pages $\pi' \preceq \pi$, and T_2 contains all the pages $\pi' \succ \pi$. Notice that the condition $|K(F)| \geq 12 \log n$ ensures that F_2 will always contain at least one page from F . All involved variables are read and written resiliently.

Overall, the **split** operation can be performed in $O(\log^2 n)$ time in the worst-case.

Merge. The **merge** operation takes two non-empty folders F_1 and F_2 such that $|F_1| + |F_2| \leq 6 \log^2 n$, $I(F_1)$ precedes $I(F_2)$, and $I(F_1) \cup I(F_2)$ is a contiguous interval, and returns a new folder F containing the keys in $K(F_1) \cup K(F_2)$ and such that $I(F) = I(F_1) \cup I(F_2)$. The operation destroys F_1 and F_2 .

We first handle some corner cases that only happen if F_1 contains a single page π . If this is the case and $|K(\pi)| \leq 2 \log n$ also holds, then we let π' be the first page of F_2 and proceed as follows:

- If $|K(\pi)| + |K(\pi')| \leq 5 \log n$, **merge** π' and π (destroying π), and return F_2 (where $I(F_2)$ and $|K(F_2)|$ are suitably updated);
- Otherwise, if $|K(\pi')| \geq 4 \log n$, we **split** π' into π_1 and π_2 , we merge π_1 and π (so that π is destroyed), and we return a new folder consisting of π_1 , π_2 and all pages other than π' in F_2 . Both π_1 and π_2 contain between $2 \log n$ and $5 \log n$ keys.
- In the remaining case, we have that $5 \log n < |K(\pi)| + |K(\pi')| < 6 \log n$. We, first **merge** π and π' , then we split π into π_1 and π_2 , each containing more than $2 \log n$ and at most $3 \log n$ keys. We return the folder consisting of π_1 , π_2 and all pages other than π' in F_2 .

We now suppose that we are not in any of the above cases, and preprocess F_1 and F_2 to ensure that the leftmost page π' of F_2 will always contain at least $2 \log n$ keys. If that is not already the case, we let π be the last page of F_1 and modify F_1 and F_2 as follows:

- If $|K(\pi)| + |K(\pi')| \leq 5 \log n$ we **merge** π' and π . We decrement $|F_1|$ to account for the destroyed page π (this might cause F_1 to temporarily become empty).
- If $|K(\pi)| \geq 4 \log n$ we first split π into π_1 and π_2 and then **merge** π_2 and π' . We store π_1 and π_2 in place of the, now destroyed, pages π and π' in F_1 and F_2 , respectively.
- Otherwise we must have $5 \log n < |K(\pi)| + |K(\pi')| < 6 \log n$. We first **merge** π and π' , then we **split** π into π_1 and π_2 . We store π_1 and π_2 in place of π and π' in F_1 and F_2 , respectively.

The rest of the operation is now straightforward: we simply create a new folder F by concatenating all pages in F_1 with those in F_2 and set $|F| = |F_1| + |F_2|$, $|K(F)| = |K(F_1)| + |K(F_2)|$, and $I(F) = I(F_1) \cup I(F_2)$. We remark that the involved pages are simply moved from their previous folder to the new folder F , rather than being destroyed and recreated anew. Overall, the merge operation requires $O(\log^2 n)$ time in the worst-case.

Amortized analysis

The following lemma, whose proof is omitted, summarizes the time complexities of the folder's operations and shows that the $O(\log^2 n)$ time needed by the expensive `insertions` and `deletions` can be amortized over $\Omega(\log n)$ non-expensive operations.

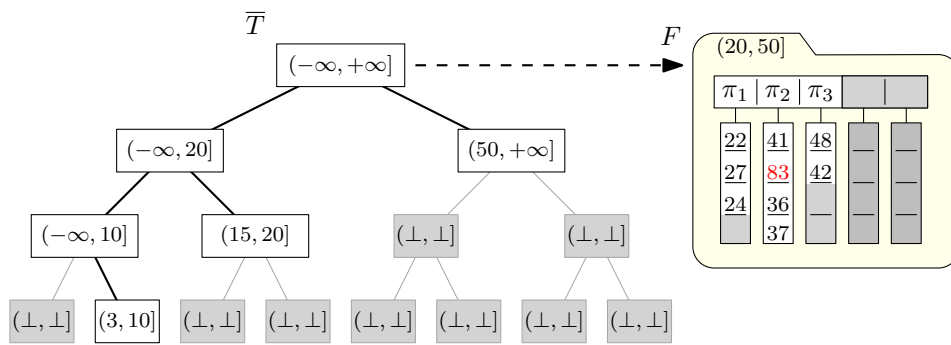
► **Lemma 6.** *An empty folder can be constructed in $O(\log n)$ worst-case time. Each search operation can be performed in $O(\log n)$ worst-case time. Merge and split operations requires $O(\log^2 n)$ worst-case time. Each insert or delete operation requires $O(\log n)$ amortized time.*

5 Our dictionary

We store the dictionary keys into $\eta = O(\frac{n}{\log^2 n})$ folders, such that (i) each folder contains at most $6 \log^2 n$ keys and (ii) all but at most one folder contain at least $\log^2 n + 1$ keys. These folders are logically organized into a dynamic *nearly-balanced* binary search tree T , in which each vertex v_F is associated with a folder F and consists of three resilient variables, namely: and a pointer containing the address of folder F , and two variables L_{v_F} and R_{v_F} defining an interval $I(v_F) = (L_{v_F}, R_{v_F}]$ where L_{v_F} (resp. R_{v_F}) is the leftmost (resp. rightmost) endpoint among of all the intervals of the folders associated with the descendants of v_F in T (including v_F itself). The intervals of the stored folders will be pairwise disjoint and will cover the whole range $(-\infty, +\infty]$. An empty dictionary consists of a single empty folder F with $I(F) = (-\infty, +\infty]$.

As shown in [7], an embedding of a *dynamic* binary search tree with $N \geq 1$ vertices into a *static* complete binary tree of height $H(N) = \lceil \log(2N + 1) \rceil - 1$ can be maintained under insertion and deletion of vertices in $O(\log^2 N)$ amortized time per operation, assuming that vertices can be accessed and copied in constant time.⁷ For the sake of completeness we briefly sketch the construction of [7]. Let $\rho(u)$ be the ratio between the number of vertices of the subtrees rooted at u in the dynamic and in the static trees, respectively. The root r will always satisfy $\frac{1}{8} \leq \rho(r) \leq \frac{1}{2}$. This is initially true when the dynamic tree consists of single vertex since $H(1) = 1$ and $\rho(r) = \frac{1}{2^{H(1)+1}-1} = \frac{1}{3}$. Whenever an insertion or deletion needs to be performed, we first check whether it will causes the value of $H(N)$ to change and, if this is the case, the operation is handled by rebuilding the updated dynamic tree in a perfectly balanced fashion. The remaining updates are as follows: insertions are performed as in an ordinary binary search tree except when they cause the height of the newly inserted vertex u to exceed $H(N)$. In this case the subtree rooted at the lowest ancestor v of u such that $\frac{1}{8} - \frac{d(v)}{16 \cdot H(N)} \leq \rho(v) \leq \frac{1}{2} + \frac{d(v)}{2 \cdot H(N)}$ is rebuilt to be perfectly-balanced (recall that $d(v)$ is the depth of vertex v). When a vertex u needs to be deleted, it is first pushed down the dynamic tree by iteratively swapping it with either its predecessor or its successor until it becomes a leaf. Then, u is deleted and the subtree rooted on the lowest vertex v that was ancestor of u and that satisfies $\frac{1}{8} - \frac{d(v)}{16 \cdot H(N)} \leq \rho(v) \leq \frac{1}{2} + \frac{d(v)}{2 \cdot H(N)}$ is rebuilt. The current number of vertices of the dynamic tree is explicitly stored (which allows to compute $\rho(r)$ in constant time) while no additional information needs to be maintained for the values $\rho(v)$ for $v \neq r$ as the complexity needed to compute them (e.g., with a DFS visit of the subtree rooted in v) is subsumed by that of the rebalancing step.

⁷ We restated the result of [7] using the standard definitions of depth and height employed in this paper. We also fixed the parameters τ_1 , γ_1 , and γ_H of [7] to $\frac{1}{2}$, $\frac{1}{8}$, and $\frac{1}{16}$ respectively.



■ **Figure 2** On the left: an example of static tree \bar{T} of height $H(\eta) = 3$ onto which is embedded the dynamic tree T consisting of the $\eta = 6$ white vertices. The label of each vertex $v \in V(T)$ is the interval $I(v)$. On the right: the folder F pointed by the root of T . In this example F contains 3 pages π_1, π_2, π_3 . Three possible intervals for the pages in F are: $I(\pi_1) = (20, 33]$, $I(\pi_2) = (33, 41]$, and $I(\pi_3) = (41, 50]$. Key 83 in π_2 is corrupted and is highlighted in red.

Since, in our case, the number of nodes η in T can be at most $1 + \lfloor \frac{n}{\log^2 n} \rfloor$, we have that $H(\eta) \leq \log(2\eta + 1) \leq 2 + \log \frac{n}{\log^2 n}$. We therefore embed T into a static tree \bar{T} which, in turn, is implicitly stored in a positional array of $2^{H(\eta)+1} - 1 \leq 8 \frac{n}{\log^2 n}$ elements. In this way, given the address of any node of \bar{T} , we can compute the addresses of its left child, right child, and parent in constant time. Overall, the space required to store \bar{T} is at most $8 \frac{n}{\log^2 n} \cdot O(\log n) = O(\frac{n}{\log n})$. Since the resilient variables of a vertex $v \in V(T)$ can be read and written in $O(\log n)$ time, we are able to insert and delete vertices from \bar{T} in $O(\log^2 \eta) \cdot O(\log n) = O(\log^3 n)$ amortized time.⁸ We distinguish the vertices v that belong in our static tree \bar{T} but not in our *logical tree* T by setting their intervals $I(v)$ to the bogus range $(\perp, \perp]$ and their pointers to some arbitrary address (see Figure 2 for an example). Notice that, when a subtree is rebalanced, the intervals of the affected nodes also need to be recomputed. The same applies to the ancestors of a deleted or newly inserted vertex. In both cases the required time complexity is $O(\log n)$ per vertex and, since the tree is nearly balanced, it is subsumed by the amortized cost of the operation. We store η , the number of keys currently in the dictionary, and the address of the array representing \bar{T} as global variables in safe memory.

Implementing the dictionary operations

Using a technique similar to the one described in Section 4 to locate a page in a folder, we are able to `locate` the folder F responsible for a given key k in our tree T (along with the corresponding node v_F).

► **Lemma 7.** *With probability at least $1 - n^{-3}$ no `locate` operation on the dictionary fails.*

Once the `locate` operation is available, the `search` operation can be easily implemented by first `locating` the (unique) folder F in T such that $k \in I(F)$ and `searching` for k in F . Moreover, we also use the `locate` operation to implement the `insert` and `delete` operations of our dictionary, together with a suitable strategy for splitting and merging the folders of

⁸ Here the amortization is performed over all insertion and deletion of vertices in \bar{T} . In the sequel we will derive an amortized bound w.r.t. insertions and deletions of keys in our dictionary.

T when they become full or contain less than $1 + \log^2 n$ keys. Due to space limitations, a formal description of these operation and the proof of the above lemma are deferred to the full version of this paper.

Space complexity

Our dictionary as described so far achieves all the bounds of Theorem 1 except for the one regarding its space complexity, which would be $O(\log^2 n + N)$ instead of the promised $O(N)$, where N is the number of keys in the dictionary. The missing range $N = o(\log^2 n)$ can be handled by lazily building the first page and the first folder of our dictionary, and by employing a suitable halving/doubling strategy [10] to ensure that their size is always linear in the number of keys stored therein. A complete proof of Theorem 1 will appear in the full version of the paper.

6 Conclusions

We presented a resilient dictionary that is able to operate correctly on all uncorrupted keys, uses the optimal amount of space, and whose operations require $O(\log n)$ amortized time. All operations are deterministic, i.e., their execution is completely determined by their input and by the observed memory contents. The same construction can also be used if satellite data is attached to the keys, in which case the `search` operation returns either the data associated with a given uncorrupted key k or that of a key that equals k following a corruption. Moreover, one can easily implement other commonly used operation on BSTs, e.g., we can report all the N stored keys in $O(N)$ worst-case time, find the predecessors or successors in $O(\log n)$ worst-case time, support range queries in $O(\log n + t)$ time where t is the size of the output, and find (resp. extract) the minimum/maximum element in $O(\log n)$ worst-case (resp. amortized) time.

Our data structure can aid in the design of other resilient algorithms and, sometimes, can even be used as a drop-in replacement of corresponding non fault-tolerant implementations. For example, when our dictionary is augmented with the `find-min` operation described above, it can substitute the heap in the classical heapsort algorithm, which then immediately solves the resilient sorting problem in our error model using $O(n \log n)$ worst-case time.

Finally, we observe that our approach can be combined with a constant replication scheme to ensure that, for any constant $c \geq 1$ of choice, at most δ/c keys are lost when δ words are corrupted. In addition, if we slightly worsen the construction time to $O(\log n)$, the time required to report the N stored keys to $O(N + \log n)$, and the the space requirements to $O(N + \log n)$, then our results also hold even when no definitely reliable memory words exist, except for $O(1)$ *temporary* registers that can only be used to hold intermediate computation results (i.e., whose contents are invalidated at the beginning/end of each dictionary operation).

References

- 1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- 2 Yonatan Aumann and Michael A. Bender. Fault Tolerant Data Structures. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 580–589, 1996. doi:10.1109/SFCS.1996.548517.
- 3 A. Bagchi. On Sorting in the Presence of Erroneous Information. *Inf. Process. Lett.*, 43(4):213–215, 1992. doi:10.1016/0020-0190(92)90203-8.

- 4 Robert S. Boyer and J. Strother Moore. MJRTY: A fast majority vote algorithm. In *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 105–118, 1991.
- 5 Mark Braverman and Elchanan Mossel. Noisy Sorting Without Resampling. In *Proceedings of the 19th Annual Symposium on Discrete Algorithms*, pages 268–276, 2008. [arXiv:0707.1051](https://arxiv.org/abs/0707.1051).
- 6 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 Lars Arge, Michael Hoffmann, and Emo Welzl, editors, *Algorithms – ESA 2007*, pages 347–358, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- 7 Gerth Stølting Brodal, Rolf Fagerberg, and Riko Jacob. Cache oblivious search trees via binary trees of small height. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, pages 39–48, 2002. URL: <http://dl.acm.org/citation.cfm?id=545381.545386>.
- 8 Paul Christiano, Erik D. Demaine, and Shaunak Kishore. Lossless Fault-Tolerant Data Structures with Additive Overhead. In *Algorithms and Data Structures - 12th International Symposium, WADS 2011, New York, NY, USA, August 15-17, 2011. Proceedings*, pages 243–254, 2011. doi:10.1007/978-3-642-22300-6_21.
- 9 Ferdinando Cicalese. *Fault-Tolerant Search Algorithms - Reliable Computation with Unreliable Information*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2013.
- 10 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: <http://mitpress.mit.edu/books/introduction-algorithms>.
- 11 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.
- 12 Irene Finocchi, Fabrizio Grandoni, and Giuseppe F. Italiano. Optimal resilient sorting and searching in the presence of memory faults. *Theor. Comput. Sci.*, 410(44):4457–4470, 2009. doi:10.1016/j.tcs.2009.07.026.
- 13 Irene Finocchi, Fabrizio Grandoni, and Giuseppe F. Italiano. Resilient dictionaries. *ACM Trans. Algorithms*, 6(1):1:1–1:19, 2009. doi:10.1145/1644015.1644016.
- 14 Irene Finocchi and Giuseppe F. Italiano. Sorting and Searching in Faulty Memories. *Algorithmica*, 52(3):309–332, 2008. doi:10.1007/s00453-007-9088-4.
- 15 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Sorting with Recurrent Comparison Errors. In Yoshio Okamoto and Takeshi Tokuyama, editors, *28th International Symposium on Algorithms and Computation (ISAAC 2017)*, volume 92 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:12, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ISAAC.2017.38.
- 16 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Optimal Dislocation with Persistent Errors in Subquadratic Time. In *Proc. of the 35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*, volume 96 of *LIPIcs*, pages 36:1–36:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.36.
- 17 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Optimal Sorting with Persistent Comparison Errors. In *Proc. of the 27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.46.
- 18 Marius Hillenbrand. Physical Address Decoding in Intel Xeon v3/v4 CPUs: A Supplemental Datasheet, 2017. URL: https://os.itec.kit.edu/downloads/publ_2017_hillenbrand_xeon_decoding.pdf.
- 19 Matthias Jung, Carl Christian Rheinländer, Christian Weis, and Norbert Wehn. Reverse Engineering of DRAMs: Row Hammer with Crosshair. In *Proceedings of the Second International Symposium on Memory Systems, MEMSYS 2016, Alexandria, VA, USA, October 3-6, 2016*, pages 471–476, 2016. doi:10.1145/2989081.2989114.
- 20 Rolf Klein, Rainer Penninger, Christian Sohler, and David P. Woodruff. Tolerant Algorithms. In *Proc. of the 19th Annual European Symposium on Algorithm (ESA)*, pages 736–747, 2011.

70:16 Resilient Dictionaries for Randomly Unreliable Memory

- 21 K. B. Lakshmanan, Bala Ravikumar, and K. Ganesan. Coping with Erroneous Information while Sorting. *IEEE Trans. Computers*, 40(9):1081–1084, 1991. doi:10.1109/12.83656.
- 22 Philip M. Long. Sorting and Searching with a Faulty Comparison Oracle. Technical report, University of California at Santa Cruz, Santa Cruz, CA, USA, 1992.
- 23 Andrzej Pelc. Searching games with errors - fifty years of coping with liars. *Theor. Comput. Sci.*, 270(1-2):71–109, 2002.
- 24 Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: exploiting DRAM addressing for cross-cpu attacks. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 565–581, 2016. URL: <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/pessl>.
- 25 Aviad Rubinfeld and Shai Vardi. Sorting from Noisier Samples. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 960–972, 2017. doi:10.1137/1.9781611974782.60.

Hermitian Laplacians and a Cheeger Inequality for the Max-2-Lin Problem

Huan Li

School of Computer Science, Fudan University, China
huanli16@fudan.edu.cn

He Sun

School of Informatics, University of Edinburgh, United Kingdom
h.sun@ed.ac.uk

Luca Zanetti

Department of Computer Science and Technology, University of Cambridge, United Kingdom
luca.zanetti@cl.cam.ac.uk

Abstract

We study spectral approaches for the MAX-2-LIN(k) problem, in which we are given a system of m linear equations of the form $x_i - x_j \equiv c_{ij} \pmod k$, and required to find an assignment to the n variables $\{x_i\}$ that maximises the total number of satisfied equations.

We consider Hermitian Laplacians related to this problem, and prove a Cheeger inequality that relates the smallest eigenvalue of a Hermitian Laplacian to the maximum number of satisfied equations of a MAX-2-LIN(k) instance \mathcal{I} . We develop an $\tilde{O}(kn^2)$ time algorithm that, for any $(1 - \varepsilon)$ -satisfiable instance, produces an assignment satisfying a $(1 - O(k)\sqrt{\varepsilon})$ -fraction of equations. We also present a subquadratic-time algorithm that, when the graph associated with \mathcal{I} is an expander, produces an assignment satisfying a $(1 - O(k^2)\varepsilon)$ -fraction of the equations. Our Cheeger inequality and first algorithm can be seen as generalisations of the Cheeger inequality and algorithm for MAX-CUT developed by Trevisan.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms; Theory of computation \rightarrow Approximation algorithms analysis; Mathematics of computing \rightarrow Spectra of graphs

Keywords and phrases Spectral methods, Hermitian Laplacians, the Max-2-Lin problem, Unique Games

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.71

Related Version A full version of the paper is available at <https://arxiv.org/abs/1811.10909>.

Funding Luca Zanetti: Supported by the ERC Starting Grant “DYNAMIC MARCH”.

Acknowledgements We are very grateful to Mihai Cucuringu for valuable discussions on Hermitian Laplacian matrices and their applications. We would also like to thank Chris Heunen for some fruitful conversations on topics closely related to this paper.

1 Introduction

In the MAX-2-LIN(k) problem, we are given a system of m linear equations of the form $u_i - v_i \equiv c_i \pmod k$, where $u_i, v_i \in \{x_1, \dots, x_n\}$ and each equation has weight b_i . The objective is to find an assignment to the variables x_i that maximises the total weight of satisfied equations. As an important case of Unique Games [8, 15], the MAX-2-LIN(k) problem has been extensively studied in theoretical computer science. This problem is known to be NP-hard to approximate within a ratio of $11/12 + \delta$ for any constant $\delta > 0$ [9, 13], and it is conjectured to be hard to distinguish between MAX-2-LIN(k) instances for which a $(1 - \varepsilon)$ -fraction of equations can be satisfied versus instances for which only an ε -fraction can



© Huan Li, He Sun, and Luca Zanetti;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 71; pp. 71:1–71:14



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

be satisfied [16]. On the algorithmic side, there has been a number of LP and SDP-based algorithms proposed for the MAX-2-LIN(k) problem (e.g., [6, 12, 15, 26]), and the case of $k = 2$, which corresponds to the classical MAX-CUT problem for undirected graphs [10, 14], has been widely studied over the past fifty years.

In this paper we investigate efficient spectral algorithms for MAX-2-LIN(k). For any MAX-2-LIN(k) instance \mathcal{I} with n variables, we express \mathcal{I} by a Hermitian Laplacian matrix $L_{\mathcal{I}} \in \mathbb{C}^{n \times n}$, and analyse the spectral properties of $L_{\mathcal{I}}$. In comparison to the well-known Laplacian matrix for undirected graphs [7], complex-valued entries in $L_{\mathcal{I}}$ are able to express directed edges in the graph associated with \mathcal{I} , and at the same time ensure that all the eigenvalues of $L_{\mathcal{I}}$ are real-valued. We demonstrate the power of our Hermitian Laplacian matrices by relating the maximum number of satisfied equations of \mathcal{I} to the spectral properties of $L_{\mathcal{I}}$. In particular, we develop a Cheeger inequality that relates partial assignments of \mathcal{I} to $\lambda_1(L_{\mathcal{I}})$, the smallest eigenvalue of $L_{\mathcal{I}}$. Based on a recursive application of the algorithm behind our Cheeger inequality, as well as a spectral sparsification procedure for MAX-2-LIN(k) instances, we present an approximation algorithm for MAX-2-LIN(k) that runs in $\tilde{O}(k \cdot n^2)$ time¹. To the best of our knowledge, this is the first purely spectral polynomial-time algorithm for the MAX-2-LIN(k) problem with approximation guarantees that matches SDP-based ones for constant values of k . The formal statement of our result is as follows:

► **Theorem 1.** *There is an $\tilde{O}(k \cdot n^2)$ -time algorithm such that, for any given MAX-2-LIN(k) instance \mathcal{I} with optimum $1 - \varepsilon$, the algorithm returns an assignment ϕ satisfying at least a $(1 - O(k)\sqrt{\varepsilon})$ -fraction of the equations².*

Our result can be viewed as a generalisation of the MAX-CUT algorithm by Trevisan [27], who derived a Cheeger inequality that relates the value of the maximum cut to the smallest eigenvalue of an undirected graph's adjacency matrix. The proof of Trevisan's Cheeger inequality, however, is based on constructing sweep sets in \mathbb{R} , while in our setting constructing sweep sets in \mathbb{C} is needed, as the underlying graph defined by $L_{\mathcal{I}}$ is directed and eigenvectors of $L_{\mathcal{I}}$ are in \mathbb{C}^n . The other difference between our result and the one in [27] is that the goal of the MAX-CUT problem is to find a *bipartition* of the vertex set, while for the MAX-2-LIN(k) problem we need to use an eigenvector to find k vertex-disjoint subsets, which corresponds to subsets of variables assigned to the same value.

Our approach also shares some similarities with the one by Goemans and Williamson [11], who presented a 0.793733-approximation algorithm for MAX-2-LIN(3) based on Complex Semidefinite Programming. The objective function of their SDP relaxation is, in fact, exactly the quadratic form of our Hermitian Laplacian matrix $L_{\mathcal{I}}$, although this matrix was not explicitly defined in their paper. In addition, their rounding scheme divides the complex unit ball into k regions according to the angle with a random vector, which is part of our rounding scheme as well. Therefore, if one views Trevisan's work [27] as a spectral analogue to the celebrated SDP-based algorithm for MAX-CUT by Goemans and Williamson [10], our result can be seen as a spectral analogue to the Goemans and Williamson's algorithm for MAX-2-LIN(k).

We further prove that, when the undirected graph associated with a MAX-2-LIN(k) instance is an expander, the approximation ratio from Theorem 1 can be improved. Our result is formally stated as follows:

¹ The notation $\tilde{O}(\cdot)$ suppresses poly-logarithmic factors in n , m , and k .

² An instance \mathcal{I} has optimum $1 - \varepsilon$, if the maximum fraction of the total weights of satisfied equations is $1 - \varepsilon$.

► **Theorem 2.** *Let \mathcal{I} be an instance of MAX-2-LIN(k) on a d -regular graph with n vertices and suppose its optimum is $1 - \varepsilon$. There is an $\tilde{O}\left(nd + \frac{n^{1.5}}{k\sqrt{\varepsilon}}\right)$ -time algorithm that returns an assignment $\phi : V \rightarrow [k]$ satisfying at least a*

$$1 - O(k^2) \cdot \frac{\varepsilon}{\lambda_2^3(\mathcal{L}_{\mathcal{U}})} \quad (1)$$

fraction of equations in \mathcal{I} , where $\lambda_2(\mathcal{L}_{\mathcal{U}})$ is the second smallest eigenvalue of the normalised Laplacian matrix of the underlying undirected graph \mathcal{U} .

Our technique is similar to the one by Kolla [18], which was used to show that solving the MAX-2-LIN(k) problem on expander graphs is easier. In [18], a MAX-2-LIN(k) instance is represented by the label-extended graph, and the algorithm is based on an exhaustive search in a subspace spanned by eigenvectors associated with eigenvalues close to 0. When the underlying graph of the MAX-2-LIN(k) instance has good expansion, this subspace is of dimension k . Therefore, the exhaustive search runs in time $O(2^k + \text{poly}(n \cdot k))$, which is polynomial-time when $k = O(\log n)$. Comparing with the work in [18], we show that, when the underlying graph has good expansion, the eigenvector associated with the smallest eigenvalue $\lambda_1(\mathcal{L}_{\mathcal{I}})$ of the Hermitian Laplacians suffices to give a good approximation. We notice that Arora et al. [4] already showed that, for expander graphs, it is possible to satisfy a $1 - O(\varepsilon \log(1/\varepsilon))$ fraction of equations in polynomial time without any dependency on k . Their algorithm is based on an SDP relaxation.

1.1 Other related work

There are many research results for the MAX-2-LIN(k) problem (e.g., [6, 12, 15, 26]), and we briefly discuss the ones most closely related to our work. For the MAX-2-LIN(k) problem and Unique Games, spectral techniques are often employed to analyse the Laplacian matrix of the Label-Extended graph (see, e.g., the aforementioned [18]), which has a strong connection with our Hermitian Laplacian: the latter can be seen as one of the blocks that arise in a particular block-diagonalisation of the former. Arora et al. [3], instead, use spectral techniques to obtain a particular decomposition of the constraint graph of a Unique Games instance, and exploit this decomposition to design an $\exp((kn)^{O(\varepsilon)}) \text{poly}(n)$ -time algorithm for Unique Games. Regarding polynomial-time algorithms, Charikar et al. [6] propose an SDP-based algorithm for Unique Games that satisfies a $1 - O(\sqrt{\varepsilon} \log k)$ fraction of constraints, which is nearly optimal assuming the Unique Games Conjecture [16]. We remark that canonical SDP programs for Unique Games can be solved in nearly-linear time [25].

Our result also relates to the research on spectral methods for synchronisation problems. For example, the adjacency matrix corresponding to our Hermitian Laplacian is considered by Singer [23] in relation to an angular synchronisation problem. The relation between the eigenvectors of such matrix and the MAX-2-LIN(k) problem is also mentioned but without offering formal approximation guarantees. Bandeira et al. [5] prove a Cheeger-type inequality that relates the spectra of an operator, the graph connection Laplacian, to “how well” an instance of the $O(d)$ -synchronisation problem can be solved. Their results, however, are not directly comparable to ours: even though our Hermitian Laplacian can also be seen as a graph connection Laplacian for an $SO(2)$ -synchronisation problem, our goal is to assign the n vertices to at most k elements of $SO(2)$, while the goal of Bandeira et al. is to assign each vertex to a possibly different element of $O(d)$.

2 Hermitian Matrices for MAX-2-LIN(k)

We can write an instance of MAX-2-LIN(k) by $\mathcal{I} = (G, k)$, where $G = (V, E, b, c)$ denotes a directed graph with an edge weight function $b : E \rightarrow \mathbb{R}^+$ and an edge color function $c : E \rightarrow [k]$, where $[k] \stackrel{\text{def}}{=} \{0, 1, \dots, k-1\}$. More precisely, every equation $u_i - v_i \equiv c_i \pmod k$ with weight b_i corresponds to a directed edge (u_i, v_i) with weight $b(u_i, v_i) = b_{u_i v_i} = b_i$ and color $c(u_i, v_i) = c_{u_i v_i} = c_i$. In the rest of this paper, we will assume that G is weakly connected, and write $u \rightsquigarrow v$ if there is a directed edge from u to v . The conjugate transpose of any vector $x \in \mathbb{C}^n$ is denoted by x^* .

We define the Hermitian adjacency matrix $A_{\mathcal{I}} \in \mathbb{C}^{n \times n}$ for instance \mathcal{I} by

$$(A_{\mathcal{I}})_{uv} \stackrel{\text{def}}{=} \begin{cases} b_{uv} \omega_k^{c_{uv}} & u \rightsquigarrow v, \\ b_{vu} \overline{\omega_k}^{c_{vu}} & v \rightsquigarrow u, \\ 0 & \text{otherwise,} \end{cases}$$

where $\omega_k = \exp\left(\frac{2\pi i}{k}\right)$ is the complex k -th root of unity, and $\overline{\omega_k} = \exp\left(-\frac{2\pi i}{k}\right)$ is its conjugate. We define the degree-diagonal matrix $D_{\mathcal{I}}$ by $(D_{\mathcal{I}})_{uu} = d_u$ where d_u is the weighted degree given by $d_u \stackrel{\text{def}}{=} \sum_{u \rightsquigarrow v} b_{uv} + \sum_{v \rightsquigarrow u} b_{vu}$. The Hermitian Laplacian matrix is then defined by $L_{\mathcal{I}} = D_{\mathcal{I}} - A_{\mathcal{I}}$, and the corresponding normalised Laplacian matrix by $\mathcal{L}_{\mathcal{I}} = D_{\mathcal{I}}^{-1/2} L_{\mathcal{I}} D_{\mathcal{I}}^{-1/2} = I - D_{\mathcal{I}}^{-1/2} A_{\mathcal{I}} D_{\mathcal{I}}^{-1/2}$. The eigenvalues of any matrix A are expressed by $\lambda_1(A) \leq \dots \leq \lambda_n(A)$. The quadratic forms of $L_{\mathcal{I}}$ can be related to the corresponding instance of MAX-2-LIN(k) by the following lemma.

► **Lemma 3.** *For any vector $x \in \mathbb{C}^n$, we have $x^* L_{\mathcal{I}} x = \sum_{u \rightsquigarrow v} b_{uv} \|x_u - \omega_k^{c_{uv}} x_v\|^2$ and*

$$x^* L_{\mathcal{I}} x = 2 \sum_{u \in V} d_u \|x_u\|^2 - \sum_{u \rightsquigarrow v} b_{uv} \|x_u + \omega_k^{c_{uv}} x_v\|^2.$$

The lemma below presents a qualitative relationship between the eigenvector associated with $\lambda_1(\mathcal{L}_{\mathcal{I}})$ and an assignment of \mathcal{I} .

► **Lemma 4.** *All eigenvalues of $\mathcal{L}_{\mathcal{I}}$ are in the range $[0, 2]$. Moreover, $\lambda_1(\mathcal{L}_{\mathcal{I}}) = 0$ if and only if there exists an assignment satisfying all equations in \mathcal{I} .*

3 A Cheeger inequality for $\lambda_1(\mathcal{L}_{\mathcal{I}})$ and MAX-2-LIN(k)

The discrete Cheeger inequality [1] shows that, for any undirected graph G , the conductance h_G of $G = (V, E)$ can be approximated by the second smallest eigenvalue of G 's normalised Laplacian matrix \mathcal{L}_G , i.e.,

$$\frac{\lambda_2(\mathcal{L}_G)}{2} \leq h_G \leq \sqrt{2 \cdot \lambda_2(\mathcal{L}_G)}. \quad (2)$$

Moreover, the proof of the second inequality above is constructive, and indicates that a subset $S \subset V$ with conductance at most $\sqrt{2 \cdot \lambda_2(\mathcal{L}_G)}$ can be found by using the second bottom eigenvector of \mathcal{L}_G to embed vertices on the real line. As one of the most fundamental results in spectral graph theory, the Cheeger inequality has found applications in the study of a wide range of optimisation problems, e.g., graph partitioning [20], max-cut [27], and many practical problems like image segmentation [22] and web search [17].

In this section, we develop connections between $\lambda_1(\mathcal{L}_{\mathcal{I}})$ and MAX-2-LIN(k) by proving a Cheeger-type inequality. Let $\phi : \{x_1, \dots, x_n\} \rightarrow [k] \cup \{\perp\}$ be an arbitrary *partial assignment* of an instance \mathcal{I} , where $\phi(x_i) = \perp$ means that the assignment of x_i has not been decided.

These variables' assignments will be determined through some recursive construction, which will be elaborated in Section 5. We remark that this framework of recursively computing a partial assignment was first introduced by Trevisan [27], and our theorem can be viewed as a generalisation of the one in [27], which corresponds to the $k = 2$ case of ours.

To relate quadratic forms of \mathcal{L}_G with the objective function of the MAX-2-LIN(k) problem, we introduce a *penalty* function as follows:

► **Definition 5.** *Given a partial assignment $\phi : \{x_1, \dots, x_n\} \rightarrow [k] \cup \{\perp\}$ and a directed edge (u, v) , the penalty of (u, v) with respect to ϕ is defined by*

$$p_{uv}^\phi(\mathcal{I}) \stackrel{\text{def}}{=} \begin{cases} 0 & \phi(u) \neq \perp, \phi(v) \neq \perp, \phi(u) - \phi(v) \equiv c_{uv} \pmod{k} \\ 1 & \phi(u) \neq \perp, \phi(v) \neq \perp, \phi(u) - \phi(v) \not\equiv c_{uv} \pmod{k} \\ 0 & \phi(u) = \phi(v) = \perp \\ 1 - \frac{1}{k} & \text{exactly one of } \phi(u), \phi(v) \text{ is } \perp. \end{cases} \quad (3)$$

For simplicity, we write p_{uv}^ϕ when the underlying instance \mathcal{I} is clear from the context.

The values of p_{uv}^ϕ from Definition 5 are chosen according to the following facts: (1) If both u and v 's values are assigned, then their penalty is 1 if the equation $\phi(u) - \phi(v) \equiv c_{uv} \pmod{k}$ associated with (u, v) is unsatisfied, and 0 otherwise; (2) If both u and v 's values are \perp , then their penalty is temporally set to 0. Their penalty will be computed when u and v 's assignment are determined during a later recursive stage. (3) If exactly one of u, v is assigned, p_{uv}^ϕ is set to $1 - 1/k$, since a random assignment to the other variable makes the edge (u, v) satisfied with probability $1/k$, hence p_{uv}^ϕ is set to $1 - 1/k$.

Without loss of generality, we only consider ϕ for which $\phi(u) \neq \perp$ for at least one vertex u , and define the penalty of assignment ϕ by

$$p^\phi \stackrel{\text{def}}{=} \frac{2 \sum_{u \rightsquigarrow v} b_{uv} p_{uv}^\phi}{\text{Vol}(\phi)}, \quad (4)$$

where $\text{Vol}(\phi) \stackrel{\text{def}}{=} \sum_{\phi(u) \neq \perp} d_u$. Notice that the p_{uv}^ϕ 's value is multiplied by b_{uv} in accordance with the objective of MAX-2-LIN(k) that maximises the total weight of satisfied assignments. Also, we multiply p_{uv}^ϕ by 2 in the numerator since edges with at least one assigned endpoint are counted at most twice in $\text{Vol}(\phi)$. Notice that, as long as G is weakly connected, $p^\phi = 0$ if and only if all edges are satisfied by ϕ and, in general, the smaller the value of p^ϕ , the more edges are satisfied by ϕ . With this in mind, we define the *imperfectness* $p(\mathcal{I})$ of \mathcal{I} to quantify how close \mathcal{I} is to an instance where all equations can be satisfied by a single assignment.

► **Definition 6.** *Given any MAX-2-LIN(k) instance $\mathcal{I} = (G, k)$, the imperfectness of \mathcal{I} is defined by $p(\mathcal{I}) \stackrel{\text{def}}{=} \min_{\phi \in ([k] \cup \{\perp\})^V \setminus \{\perp\}^V} p^\phi$.*

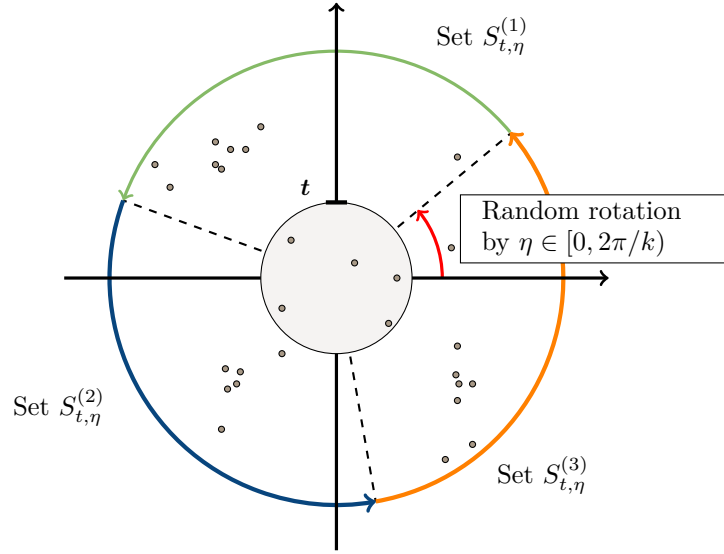
The main result of this section is a Cheeger-type inequality that relates $p(\mathcal{I})$ and $\lambda_1(\mathcal{L}_\mathcal{I})$, which is summarised in Theorem 7. Note that, since $\sin(x) \geq (2/\pi) \cdot x$ for $x \in [0, \pi/2]$, the factor before $\sqrt{2\lambda_1}$ in the theorem statement is at most $(2 + k/4)$ for $k \geq 2$.

► **Theorem 7.** *Let λ_1 be the smallest eigenvalue of $\mathcal{L}_\mathcal{I}$. It holds that*

$$\frac{\lambda_1}{2} \leq p(\mathcal{I}) \leq \left(2 - \frac{2}{k} + \frac{1}{2 \sin(\pi/k)} \right) \sqrt{2\lambda_1}. \quad (5)$$

Moreover, given the eigenvector associated with λ_1 , there is an $O(m + n \log n)$ -time algorithm that returns a partial assignment ϕ such that

$$\frac{\lambda_1}{2} \leq p^\phi \leq \left(2 - \frac{2}{k} + \frac{1}{2 \sin(\pi/k)} \right) \sqrt{2\lambda_1}. \quad (6)$$



■ **Figure 1** Illustration of the proof for Theorem 7 for the case of $k = 3$. The gray circle is obtained by sweeping $t \in [0, 1]$, and the red arrow represents a random angle $\eta \in [0, 2\pi/k)$. A partial assignment is determined by the values of η and t .

Proof Sketch of Theorem 7. We present an overview of the proof here, and a complete proof of the theorem can be found in the full version of the paper. The easy direction of (5), i.e., $\lambda_1/2 \leq p(\mathcal{I})$, follows from the Courant-Fischer characterisation of eigenvalues and that the eigenvector problem is a relaxation of $\text{MAX-2-LIN}(k)$. Hence, we will mainly sketch the techniques used to prove the other direction of (5). We assume that $z \in \mathbb{C}^n$ is the vector such that

$$\frac{z^* L_{\mathcal{I}} z}{z^* D_{\mathcal{I}} z} = \lambda_1,$$

and prove the existence of an assignment ϕ based on z satisfying

$$p^\phi \leq \left(2 - \frac{2}{k} + \frac{1}{2 \sin(\pi/k)}\right) \sqrt{2\lambda_1} = \left(2 - \frac{2}{k} + \frac{1}{2 \sin(\pi/k)}\right) \sqrt{2 \cdot \frac{z^* L_{\mathcal{I}} z}{z^* D_{\mathcal{I}} z}}.$$

We first scale each coordinate of z such that $\max_{u \in V} \|z_u\|^2 = 1$. In this way z can be seen as an embedding of the vertices to the complex unit ball. For any real numbers $t \geq 0$ and $\eta \in [0, \frac{2\pi}{k})$, we define k sets of vertices indexed by $j \in [k]$ as follows:

$$S_{t,\eta}^{(j)} = \left\{ u \mid \|z_u\| \geq t \text{ and } \theta(z_u, e^{i\eta}) \in \left[j \cdot \frac{2\pi}{k}, (j+1) \cdot \frac{2\pi}{k} \right) \right\}.$$

Here, we use $\theta(a, b) \in [-\pi, \pi)$ to represent the angle from $b \in \mathbb{C}$ to $a \in \mathbb{C}$, i.e., $\frac{a}{\|a\|} = \frac{b}{\|b\|} \exp(i\theta(a, b))$. We then define an assignment $\phi_{t,\eta}$ where $\phi_{t,\eta}(u) = j$ if there is $j \in [k]$ such that $u \in S_{t,\eta}^{(j)}$, and $\phi_{t,\eta}(u) = \perp$ otherwise. By definition, the k vertex sets correspond to the vectors in the k regions of the unit ball after each vector is rotated by η radians counterclockwise. The role of t is to only consider the coordinates z_u with $\|z_u\| \geq t$. This is illustrated in Figure 1.

Now we assume $t \in [0, 1]$ is chosen such that t^2 follows from a uniform distribution over $[0, 1]$, and η is chosen uniformly at random from $[0, 2\pi/k)$. Further calculations show that

$$\mathbb{E}_{t,\eta} [\text{Vol}(\phi_{t,\eta})] = \sum_{u \in V} d_u \cdot \mathbb{P} [\|z_u\| \geq t] = \sum_{u \in V} d_u \|z_u\|^2 = z^* D_{\mathcal{I}} z,$$

and

$$\mathbb{E}_{t,\eta} \left[2 \sum_{u \rightsquigarrow v} b_{uv} p_{uv}^\phi \right] \leq \left(2 - \frac{2}{k} + \frac{1}{2 \sin(\pi/k)} \right) \cdot \sqrt{z^* L_{\mathcal{I}} z} \cdot \sqrt{2 z^* D_{\mathcal{I}} z}.$$

Hence, it holds that

$$\frac{\mathbb{E}_{t,\eta} [2 \sum_{u \rightsquigarrow v} b_{uv} p_{uv}^\phi]}{\mathbb{E}_{t,\eta} [\text{Vol}(\phi_{t,\eta})]} \leq \left(2 - \frac{2}{k} + \frac{1}{2 \sin(\pi/k)} \right) \cdot \sqrt{2 \cdot \frac{z^* L_{\mathcal{I}} z}{z^* D_{\mathcal{I}} z}}.$$

This implies by linearity of expectation that

$$\mathbb{E}_{t,\eta} \left[2 \sum_{u \rightsquigarrow v} b_{uv} p_{uv}^\phi - \left(2 - \frac{2}{k} + \frac{1}{2 \sin(\pi/k)} \right) \cdot \text{Vol}(\phi_{t,\eta}) \cdot \sqrt{2 \cdot \frac{z^* L_{\mathcal{I}} z}{z^* D_{\mathcal{I}} z}} \right] \leq 0,$$

and existence of an assignment ϕ satisfying (6).

Now we turn to the runtime needed to find such a vertex set. Notice that we need to find t and η such that $\phi_{t,\eta}$ satisfies (6). Therefore, we construct two sequences of *sweep sets*: the first is based on t , and the second is based on η . For constructing the sweep sets based on t , the algorithm increases t from 0 to 1, and updates the corresponding conditional expectation looking only at the edges incident with u whenever t exceeds $\|z_u\|$. Notice that each edge (u, v) will be updated at most twice, i.e., the step when t reaches $\|z_u\|$ and the step when t reaches $\|z_v\|$, and the total runtime needed to update $\text{Vol}(\phi_{t,\eta})$ is $O(m)$. Hence, the total runtime for constructing the sweep sets based on t is $O(m)$. The runtime analysis for constructing the sweep sets based on η is similar: the algorithm increases η from 0 to $2\pi/k$, and updates the penalties p_{uv}^ϕ of the edges (u, v) only if the assignment of u or v changes. Since every edge will be updated at most twice, the runtime for constructing the sweep sets based on η is $O(m)$ as well. Hence, the total runtime of the algorithm is $O(m + n \log n)$. ◀

► **Remark 8.** We remark that the factors $\lambda_1/2$ and $\sqrt{\lambda_1}$ in Theorem 7 are both tight within constant factors. The tightness can be derived directly from Section 5 of [27], since when $k = 2$, our inequality is the same as the one in [27] up to constant factors.

We also remark that the factor of k in Theorem 7 is necessary, which is shown by the following instance: the linear system has nk variables where every variable belongs to one of k sets S_0, \dots, S_{k-1} with $|S_i| = n$ for any $0 \leq i \leq k-1$. Now, for any i , we add n equations of the form $x_u - x_v = 1 \pmod k$ with $x_u \in S_i$, $x_v \in S_j$, and $j = i + 1 \pmod k$, and n equations of the form $x_u - x_v = 1 \pmod k$ with $x_u \in S_i$, $x_v \in S_j$, and $j = i + 2 \pmod k$. This instance is constructed such that the underlying graph is regular, and every assignment could only satisfy at most half of the equations, implying that the imperfectness is $p(\mathcal{I}) = \Omega(1)$. However, mapping each variable in S_i to the root of unity ω_k^i , it's easy to see that $\lambda_i(\mathcal{L}_{\mathcal{I}}) = O(1/k)$. Hence Theorem 7 is tight with respect to k .

Finally, we compare the proof techniques of Theorem 7 with other Cheeger-type inequalities in the literature: first of all, most of the Cheeger-type inequalities (e.g., [1, 19, 20, 27]) consider the case where every eigenvector is in \mathbb{R}^n and are only applicable for undirected graphs, while for our problem the graph G associated with \mathcal{I} is directed and eigenvectors of

$A_{\mathcal{I}}$ are in \mathbb{C}^n . Therefore, constructing sweep sets in \mathbb{C} is needed, which is more involved than proving similar Cheeger-type inequalities (e.g., [1, 27]). Secondly, by dividing the complex unit ball into k regions, we are able to show that a partial assignment corresponding to k disjoint subsets can be found using a single eigenvector. This is quite different from the techniques used for finding k vertex-disjoint subsets of low conductance in an undirected graph, where k eigenvectors are usually needed (e.g. [19, 20, 21]).

We also remark that, while sweeping through values of t is needed to obtain *any* guarantee on the penalty of the partial assignment computed, we could in principle just choose a random angle η : in this way, however, the partial assignment returned would satisfy (6) only in expectation.

4 Sparsification for MAX-2-LIN(k)

We have seen in Section 3 that, given any vector in \mathbb{C}^n whose quadratic form with $L_{\mathcal{I}}$ is close to $\lambda_1(L_{\mathcal{I}})$, we can compute a partial assignment of \mathcal{I} with bounded approximation guarantee. In Section 5 we will show that a total assignment can be found by recursively applying this procedure on variables for which an assignment has not yet been fixed. In particular, we will show that every iteration takes a time nearly-linear in the number of equations of our instance, which can be quadratic in the number of variables. To speed-up each iteration and obtain a time per iteration that is nearly-linear in the number of variables, we need to sparsify our input instance \mathcal{I} .

In this section we show that the construction of spectral sparsifiers by effective resistance sampling introduced in [24] can be generalised to sparsify MAX-2-LIN(k) instances. In particular, given an instance \mathcal{I} of MAX-2-LIN(k) with n variables and m equations, we can find in nearly-linear time a sparsified instance \mathcal{J} with about $nk \log(nk)$ equations such that for any partial assignment $\phi : V \rightarrow [k]$, the number of unsatisfied equations in \mathcal{J} is preserved within a constant factor. This means that we can apply our algorithm for MAX-2-LIN(k) to a sparsified instance \mathcal{J} , and any dependency on m in our runtime can be replaced by $nk \log(nk)$. We remark that we could simply apply uniform sampling to obtain a sparsified instance. However, this would in the end result in an additive error in the fraction of unsatisfied equations, much like in the case of the original Trevisan's result for MAX-CUT [27]. With our construction, instead, we only lose a small multiplicative error.

To construct a sparsified instance \mathcal{J} , we introduce label-extended graphs and their Laplacian matrices to characterise the original MAX-2-LIN(k) instance. Let $P \in \mathbb{R}^{k \times k}$ be the permutation matrix where $P_{ij} = 1$ if $i \equiv j + 1 \pmod k$, and $P_{ij} = 0$ otherwise. We define the adjacency matrix $\tilde{A}_{\mathcal{I}} \in (\mathbb{R}^{k \times k})^{n \times n}$ for the label-extended graph of instance \mathcal{I} , where each entry of $\tilde{A}_{\mathcal{I}}$ is a matrix in $\mathbb{R}^{k \times k}$ given by

$$(\tilde{A}_{\mathcal{I}})_{uv} \stackrel{\text{def}}{=} \begin{cases} b_{uv} P^{c_{uv}} & u \rightsquigarrow v, \\ b_{vu} (P^{\top})^{c_{vu}} & v \rightsquigarrow u, \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

We then define the degree-diagonal matrix $\tilde{D}_{\mathcal{I}} \in (\mathbb{R}^{k \times k})^{n \times n}$ by $(\tilde{D}_{\mathcal{I}})_{uu} = d_u \cdot I_{k \times k}$, where $I_{k \times k}$ is the $k \times k$ identity matrix, and define the Laplacian matrix by $\tilde{L}_{\mathcal{I}} = \tilde{D}_{\mathcal{I}} - \tilde{A}_{\mathcal{I}}$. Notice that the Hermitian Laplacian $\mathcal{L}_{\mathcal{I}}$ is a *compression* of $\tilde{L}_{\mathcal{I}}$, i.e., there exists an orthogonal projection U such that $U^* \tilde{L}_{\mathcal{I}} U = \mathcal{L}_{\mathcal{I}}$.

For any assignment $\phi : V \rightarrow [k]$, we construct an indicator vector $\tilde{x}_{\mathcal{I}} \in (\mathbb{R}^k)^n$ by $(\tilde{x}_{\mathcal{I}})_u = e_{\phi(u)}$, where $e_j \in \mathbb{R}^k$ is the j -th standard basis vector. Then, it is easy to show that the total weight of unsatisfied equations for ϕ is $(1/2) \cdot \tilde{x}_{\mathcal{I}}^\top \tilde{L}_{\mathcal{I}} \tilde{x}_{\mathcal{I}}$.³

We show that, for every unsatisfiable instance \mathcal{I} , there is a sparsified MAX-2-LIN(k) instance \mathcal{J} such that the quadratic forms between $\tilde{L}_{\mathcal{I}}$ and $\tilde{L}_{\mathcal{J}}$ are approximately preserved. This implies that, when looking at the same assignment, the total weights of unsatisfied equations in \mathcal{I} and \mathcal{J} are approximately preserved. Notice that we can decide whether there is an assignment satisfying all the equations in \mathcal{I} by fixing the assignment of an arbitrary vertex and determining assignments for other vertices accordingly, and therefore we only need to consider the case when \mathcal{I} is unsatisfiable. The main result of the section is as follows:

► **Theorem 9.** *There is an algorithm that, given an unsatisfiable instance \mathcal{I} of MAX-2-LIN(k) with n variables and m equations and parameter $0 < \delta < 1$, returns in $\tilde{O}(mk)$ time an instance \mathcal{J} with the same set of variables and $O((1/\delta^2) \cdot nk \log(nk))$ equations. Furthermore, with high probability it holds for any vector $x \in (\mathbb{R}^k)^n$ that $(1-\delta)x^\top \tilde{L}_{\mathcal{I}} x \leq x^\top \tilde{L}_{\mathcal{J}} x \leq (1+\delta)x^\top \tilde{L}_{\mathcal{I}} x$.*

5 Algorithm for MAX-2-LIN(k)

Theorem 9 tells us that, given an instance \mathcal{I}^* , we can find a sparse instance \mathcal{I} so that the quadratic forms of the corresponding Laplacians $\mathcal{L}_{\mathcal{I}^*}$ and $\mathcal{L}_{\mathcal{I}}$ are approximately the same. Therefore throughout this section we assume that the input instance \mathcal{I} for MAX-2-LIN(k) with n variables has $m = \tilde{O}((1/\delta^2) \cdot nk)$ equations for some parameter $\delta > 0$. Recall that Theorem 7 shows that, for any MAX-2-LIN(k) instance \mathcal{I} , given an eigenvector for the smallest eigenvalue $\lambda_1(\mathcal{L}_{\mathcal{I}})$, we can obtain a partial assignment ϕ satisfying

$$p^\phi \leq \left(2 - \frac{2}{k} + \frac{1}{2 \sin(\pi/k)}\right) \sqrt{2\lambda_1}. \quad (8)$$

Now we show that, by a repeated application of Theorem 7 on the subset of the equations of \mathcal{I} for which both variables are unassigned, we can obtain a full assignment of \mathcal{I} . Our algorithm closely follows the one by Trevisan [27] and is described in Algorithm 1.

To achieve the guarantees of (8), however, we would need to compute the eigenvector corresponding to $\lambda_1(\mathcal{L}_{\mathcal{I}})$ *exactly*. To obtain a nearly-linear time algorithm, instead, we relax this requirement and compute a vector z that well-approximates this eigenvector. In particular, the following lemma shows that, for any δ , we can compute a vector $z \in \mathbb{C}^n$ satisfying (9) in nearly-linear time.

► **Lemma 10.** *For any given error parameter δ , there is an $\tilde{O}((1/\delta^3) \cdot kn)$ time algorithm that returns $z \in \mathbb{C}^n$ satisfying (9).*

To analyse Algorithm 1, we introduce some notation. Let t be the number of recursive executions of Algorithm 1. For any $1 \leq j \leq t+1$, let \mathcal{I}_j be the instance of MAX-2-LIN(k) in the j -th execution. We indicate with $\rho_j m$ the number of equations in \mathcal{I}_j , where $0 \leq \rho_j \leq 1$. Notice that $\mathcal{I}_1 = \mathcal{I}$ and $\mathcal{I}_{t+1} = \emptyset$. We assume that the maximum number of equations in \mathcal{I}_j that can be satisfied by an assignment is $(1 - \varepsilon_j)\rho_j m$, with $\varepsilon = \varepsilon_1$. Also notice that it holds for any $1 \leq j \leq t$ that $\varepsilon_j \rho_j m \leq \varepsilon m$, which implies $\varepsilon_j \leq \varepsilon/\rho_j$. The next theorem presents the performance of our algorithm, whose informal version is Theorem 1.

³ We remark that, if we use the Hermitian Laplacian matrices $L_{\mathcal{I}}$ directly instead, this relation only holds up to an $O(k)$ factor. That is why we sparsify the matrix $\tilde{L}_{\mathcal{I}}$ instead.

■ **Algorithm 1** RECURSIVECONSTRUCT(\mathcal{I}, δ).

1: Compute vector $z \in \mathbb{C}^n$ satisfying

$$\frac{z^* L_{\mathcal{I}} z}{z^* D_{\mathcal{I}} z} \leq (1 + 2\delta) \lambda_1(\mathcal{L}_{\mathcal{I}}); \quad (9)$$

2: Apply the algorithm from Theorem 7 to compute $\phi : V \rightarrow [k] \cup \{\perp\}$ such that

$$p^\phi \leq (1 + \delta) \left(2 - \frac{2}{k} + \frac{1}{2 \sin(\pi/k)} \right) \sqrt{2\lambda_1}; \quad (10)$$

3: **if** $2p^\phi \geq (1 - 1/k) \text{Vol}(\phi)$ **then**

4: return random full assignment $\phi' : V \rightarrow [k]$;

5: ▷ the case where the current assignment is worse than a random assignment

6: **else if** ϕ is a full assignment (i.e. $\phi(V) \subseteq [k]$) **then**

7: return ϕ ;

8: ▷ The recursion terminates if every variable's assignment is determined

9: **else**

10: $\mathcal{I}' \leftarrow$ set of equations from \mathcal{I} in which both variables' assignments are not determined;

11: **if** $\mathcal{I}' = \emptyset$ **then**

12: set $\phi(u)$ to be an arbitrary assignment if $\phi(u) = \perp$ for any u ;

13: return ϕ ;

14: **else**

15: $\phi_1 \leftarrow$ RECURSIVECONSTRUCT(\mathcal{I}', δ);

16: return $\phi \cup \phi_1$;

► **Theorem 11.** *Given an instance \mathcal{I} of MAX-2-LIN(k) whose optimum is $1 - \varepsilon$ and a parameter $\delta > 0$, the algorithm RECURSIVECONSTRUCT(\mathcal{I}, δ) returns in $\tilde{O}((1/\delta^3) \cdot kn^2)$ time an assignment ϕ satisfying at least $1 - 8\nu\sqrt{\varepsilon}$ fraction of the equations, where*

$$\nu \stackrel{\text{def}}{=} (1 + \delta) \left(2 - \frac{2}{k} + \frac{1}{2 \sin(\pi/k)} \right) = O(k).$$

The following corollary which states how much our algorithm beats a random assignment follows from Theorem 1.

► **Corollary 12.** *Given a MAX-2-LIN(k) instance \mathcal{I} whose optimum is ξ and a constant $\delta > 0$, Algorithm 1 returns in $\tilde{O}((1/\delta^3) \cdot n^2 k)$ time an assignment ϕ satisfying at least $(1/k + \tau)\xi$ fraction of the equations, where $\tau = \Omega(\frac{1}{k^3})$.*

6 Algorithm for MAX-2-LIN(k) on expanders

In this section we further develop techniques for analysing Hermitian Laplacian matrices by presenting a subquadratic-time approximation algorithm for the MAX-2-LIN(k) problem on expander graphs. Our proof technique is inspired by Kolla's algorithm [18]. However, in contrast to the algorithm in [18], we use the Hermitian Laplacian to represent a MAX-2-LIN(k) instance and show that, when the underlying graph has good expansion, a good approximate solution is encoded in the eigenvector associated with $\lambda_1(\mathcal{L}_{\mathcal{I}})$. We assume that G is a d -regular graph, and hence $\mathcal{I} = (G, k)$ is a MAX-2-LIN(k) instance with n variables and $nd/2$

equations whose optimum is $1 - \varepsilon$. One can view \mathcal{I} as an instance generated by modifying ε fraction of the constraints (i.e., edges) from a completely satisfiable instance $\widehat{\mathcal{I}} = (\widehat{G}, k)$. Hence, a satisfiable assignment $\psi : V \rightarrow [k]$ for $\widehat{\mathcal{I}}$ will satisfy at least a $(1 - \varepsilon)$ -fraction of equations in \mathcal{I} .

Now we discuss the techniques used to prove Theorem 2. Let $y_\psi \in \mathbb{C}^n$ be the normalised “indicator vector” of ψ , i.e., $(y_\psi)_u = \frac{1}{\sqrt{n}} \omega_k^{\psi(u)}$. Then it holds that

$$(y_\psi)^* \mathcal{L}_{\widehat{\mathcal{I}}} y_\psi = \frac{1}{d} \sum_{u \rightsquigarrow v} b_{uv} \|(y_\psi)_u - \omega_k^{c_{uv}} (y_\psi)_v\|^2 = 0.$$

Hence y_ψ is an eigenvector associated with $\lambda_1(\mathcal{L}_{\widehat{\mathcal{I}}}) = 0$. We denote by \mathcal{U} the underlying undirected graph of G , and denote by $\mathcal{L}_{\mathcal{U}}$ the normalised Laplacian of \mathcal{U} . Note that since \mathcal{U} is undirected, $\mathcal{L}_{\mathcal{U}}$ only contains real-valued entries. We first show that the eigenvalues of $\mathcal{L}_{\widehat{\mathcal{I}}}$, the normalised Laplacian of the completely satisfiable instance, and of $\mathcal{L}_{\mathcal{U}}$, the normalised Laplacian of the underlining undirected graph \mathcal{U} , coincide. Since $\mathcal{L}_{\mathcal{U}}$ is the Laplacian matrix of an expander graph, this implies that there is a gap between $\lambda_1(\mathcal{L}_{\widehat{\mathcal{I}}})$ and $\lambda_2(\mathcal{L}_{\widehat{\mathcal{I}}})$.

► **Lemma 13.** *It holds for all $1 \leq i \leq n$ that $\lambda_i(\mathcal{L}_{\widehat{\mathcal{I}}}) = \lambda_i(\mathcal{L}_{\mathcal{U}})$.*

Next we bound the perturbation of the bottom eigenspace of $\mathcal{L}_{\widehat{\mathcal{I}}}$ when the latter is turned into $\mathcal{L}_{\mathcal{I}}$. In particular, Lemma 14 below proves that this perturbation does not affect too much to the vectors that have norm spreads out uniformly over all their coordinates.

► **Lemma 14.** *Let $f \in \mathbb{C}^n$ be a vector such that $\|f_u\| = \frac{1}{\sqrt{n}}$ for all $u \in V$. It holds that*

$$\left\| (\mathcal{L}_{\mathcal{I}} - \mathcal{L}_{\widehat{\mathcal{I}}}) f \right\| \leq 2\sqrt{\varepsilon}. \tag{11}$$

Based on Lemma 14, we prove that the change from $\mathcal{L}_{\widehat{\mathcal{I}}}$ to $\mathcal{L}_{\mathcal{I}}$ doesn't have too much influence on the eigenvector associated with $\lambda_1(\mathcal{L}_{\mathcal{I}})$. For simplicity, let $\lambda_2 = \lambda_2(\mathcal{L}_{\widehat{\mathcal{I}}}) = \lambda_2(\mathcal{L}_{\mathcal{U}})$.

► **Lemma 15.** *Let $f_1 \in \mathbb{C}^n$ be a unit eigenvector associated with $\lambda_1(\mathcal{L}_{\mathcal{I}})$. Then we have $\left\| (\mathcal{L}_{\mathcal{I}} - \mathcal{L}_{\widehat{\mathcal{I}}}) f_1 \right\| \leq 20\sqrt{\varepsilon/\lambda_2}$.*

We then prove the following lemma which shows that the eigenvector f_1 corresponding to $\lambda_1(\mathcal{L}_{\mathcal{I}})$ is close to y_ψ , the indicator vector of the optimal assignment ψ .

► **Lemma 16.** *Let $f_1 \in \mathbb{C}^n$ be a unit eigenvector associated with $\lambda_1(\mathcal{L}_{\mathcal{I}})$. Then, there exist $\alpha, \beta \in \mathbb{C}$ and a unit vector $y_\perp \in \mathbb{C}^n$ orthogonal to y_ψ (i.e. $(y_\perp)^* y_\psi = 0$) such that $f_1 = \alpha y_\psi + \beta y_\perp$ and $\|\beta\| \leq 30\sqrt{\varepsilon/\lambda_2^3}$.*

Based on Lemma 16, f_1 is close to the indicator vector of an optimal assignment rotated by some angle. In particular, we have that

$$\left\| f_1 - \frac{\alpha}{\|\alpha\|} y_\psi \right\| = \sqrt{(1 - \|\alpha\|)^2 + \|\beta\|^2} \leq \sqrt{1 - \|\alpha\|^2 + \|\beta\|^2} = \sqrt{2} \|\beta\| \leq 30\sqrt{\frac{2\varepsilon}{\lambda_2^3}}, \tag{12}$$

where $\frac{\alpha}{\|\alpha\|} y_\psi$ is the vector that encodes the information of an assignment that satisfies all the equations in $\widehat{\mathcal{I}}$ and at least $1 - \varepsilon$ fraction of equations in \mathcal{I} . Therefore, our goal is to recover $\frac{\alpha}{\|\alpha\|} y_\psi$ from f_1 .

71:12 Hermitian Laplacians and a Cheeger Inequality for the Max-2-Lin Problem

Proof of Theorem 2. Let ψ be the optimal assignment of \mathcal{I} satisfying $1 - \varepsilon$ fraction of equations, which is also a completely satisfying assignment of $\widehat{\mathcal{I}}$. Let f_1 be a unit eigenvector associated with $\lambda_1(\mathcal{L}_{\mathcal{I}})$. By Lemma 16, there exists $\alpha, \beta \in \mathbb{C}$ such that $f_1 = \alpha y_\psi + \beta y_\perp$ where $\|\beta\| \leq 30\sqrt{\varepsilon/\lambda_2^3}$. Our goal is to find a vector $z_\phi \in \mathbb{C}^n$, which equals the indicator vector of ϕ rotated by some angle and satisfies

$$\|f_1 - z_\phi\| \leq \left\| f_1 - \frac{\alpha}{\|\alpha\|} y_\psi \right\| \leq 30\sqrt{\frac{2\varepsilon}{\lambda_2^3}}, \quad (13)$$

where the last inequality follows by (12). The assignment ϕ corresponding to such a z_ϕ will give us that the fraction of unsatisfied equations by ϕ is

$$\begin{aligned} p^\phi(\mathcal{I}) &\leq 10k^2 z_\phi^* \mathcal{L}_{\mathcal{I}} z_\phi \\ &= 10k^2 (z_\phi - f_1 + f_1)^* \mathcal{L}_{\mathcal{I}} (z_\phi - f_1 + f_1) \\ &\leq k^2 ((z_\phi - f_1)^* \mathcal{L}_{\mathcal{I}} (z_\phi - f_1) + f_1^* \mathcal{L}_{\mathcal{I}} f_1 + 2\|(z_\phi - f_1)^* \mathcal{L}_{\mathcal{I}} f_1\|) \\ &\leq 10k^2 \left(2\|z_\phi - f_1\|^2 + \lambda_1(\mathcal{L}_{\mathcal{I}}) + 2\|z_\phi - f_1\| \sqrt{\lambda_1(\mathcal{L}_{\mathcal{I}})} \right) \\ &\leq 10k^2 \left(2 \cdot 900 \cdot \frac{2\varepsilon}{\lambda_2^3} + 2\varepsilon + 2 \cdot 30 \cdot \sqrt{\frac{2\varepsilon}{\lambda_2^3}} \cdot \sqrt{2\varepsilon} \right) \\ &\leq 100000k^2 \cdot \frac{\varepsilon}{\lambda_2^3}, \end{aligned}$$

where the factor $10k^2$ above follows from the fact that $\|1 - \omega_k^j\|^2$ is at least $1/(10k^2)$ for $j = 1, \dots, k-1$.

To find such vector z_ϕ satisfying (13), we define $\phi_\eta(u) = \arg \min_{j \in [k]} \|(f_1)_u - e^{\eta i} \omega_k^j\|$. Notice that, since $\frac{\alpha}{\|\alpha\|}$ is equal to $e^{\eta i}$ for some $\eta \in [0, 2\pi)$, by defining $(z_{\phi_\eta})_u = e^{\eta i} \omega_k^{\phi_\eta(u)}$ the solution to the following optimisation problem $\min_{\eta \in [0, 2\pi)} \|z_{\phi_\eta} - f_1\|$ gives us a vector that satisfies (13). To solve this optimisation problem, we notice that it suffices to consider η in the range $[0, 2\pi/k)$. Therefore, we simply enumerate all η 's over the following discrete set:

$$\left\{ \frac{t\sqrt{\varepsilon}}{\sqrt{n}} \mid t = 0, 1, \dots, \left\lceil \frac{2\pi\sqrt{n}}{k\sqrt{\varepsilon}} \right\rceil \right\}.$$

By enumerating this set, we can find an assignment ϕ and an η such that

$$\|f_1 - z_{\phi_\eta}\| \leq \left\| f_1 - \frac{\alpha}{\|\alpha\|} y_\psi \right\| + O(\sqrt{\varepsilon}),$$

which is enough to get our desired approximation. Since the size of this set is $O\left(\frac{\sqrt{n}}{k\sqrt{\varepsilon}}\right)$, the total running time is $O\left(\frac{n^{1.5}}{k\sqrt{\varepsilon}}\right)$ plus the running time needed to compute f_1 . \blacktriangleleft

7 Concluding remarks

Our work leaves several open questions for further research: while the factor of k in our Cheeger inequality (Theorem 7) is needed, it would be interesting to see if it's possible to construct a different Laplacian for which a similar Cheeger inequality holds with a smaller dependency on k . For example, instead of embedding vertices in \mathbb{C} and mapping assignments to roots of unity, one could consider embedding vertices in higher dimensions using the

bottom k eigenvectors of the Laplacian of the label extended graph, and see if a relation between the imperfectness ratio of Definition 6 and the k -th smallest eigenvalue of this Laplacian still holds.

Finally, we observe that several cut problems in directed graphs can be formulated as special cases of MAX-2-LIN(k) (see, e.g., [2, 11]). Because of this, we believe the Hermitian Laplacians studied in our paper will have further applications in the development of fast algorithms for combinatorial problems on directed graphs, and might have further connections to Unique Games.

References

- 1 Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.
- 2 Gunnar Andersson, Lars Engebretsen, and Johan Håstad. A New Way of Using Semidefinite Programming with Applications to Linear Equations mod p . *Journal of Algorithms*, 39(2):162–204, 2001.
- 3 Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential Algorithms for Unique Games and Related Problems. *Journal of the ACM*, 62(5), 2015.
- 4 Sanjeev Arora, Subhash Khot, Alexandra Kolla, David Steurer, Madhur Tulsiani, and Nish-eeth K. Vishnoi. Unique games on expanding constraint graphs are easy. In *40th Annual ACM Symposium on Theory of Computing (STOC'08)*, pages 21–28, 2008.
- 5 Afonso S. Bandeira, Amit Singer, and Daniel A. Spielman. A Cheeger inequality for the graph connection Laplacian. *SIAM J. Matrix Anal. Appl.*, 34(4):1611–1630, 2013.
- 6 Moses Charikar, Konstantin Makarychev, and Yury Makarychev. Near-optimal algorithms for unique games. In *38th Annual ACM Symposium on Theory of Computing (STOC'06)*, pages 205–214, 2006.
- 7 Fan R. K. Chung. Spectral Graph Theory. *Regional Conference Series in Mathematics, American Mathematical Society*, 92:1–212, 1997.
- 8 Uriel Feige and László Lovász. Two-prover one-round proof systems: Their power and their problems. In *24th Annual ACM Symposium on Theory of Computing (STOC'92)*, pages 733–744, 1992.
- 9 Uriel Feige and Daniel Reichman. On systems of linear equations with two variables per equation. In *7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'04)*, pages 117–127, 2004.
- 10 Michel X. Goemans and David P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *Journal of the ACM*, 42(6):1115–1145, 1995.
- 11 Michel X. Goemans and David P. Williamson. Approximation algorithms for Max-3-Cut and other problems via complex semidefinite programming. *Journal of Computer and System Sciences*, 68(2):442–470, 2004.
- 12 Anupam Gupta and Kunal Talwar. Approximating unique games. In *17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06)*, pages 99–106, 2006.
- 13 Johan Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.
- 14 Richard M. Karp. Reducibility Among Combinatorial Problems. In *a symposium on the Complexity of Computer Computations*, pages 85–103, 1972.
- 15 Subhash Khot. On the power of unique 2-prover 1-round games. In *34th Annual ACM Symposium on Theory of Computing (STOC'02)*, pages 767–775, 2002.
- 16 Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal Inapproximability Results for MAX-CUT and Other 2-Variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007.
- 17 Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM*, 46(5):604–632, 1999.

- 18 Alexandra Kolla. Spectral algorithms for unique games. *Computational Complexity*, 20(2):177–206, 2011.
- 19 Tsz Chiu Kwok, Lap Chi Lau, Yin Tat Lee, Shayan Oveis Gharan, and Luca Trevisan. Improved Cheeger’s inequality: analysis of spectral partitioning algorithms through higher order spectral gap. In *45th Annual ACM Symposium on Theory of Computing (STOC’13)*, pages 11–20, 2013.
- 20 James R. Lee, Shayan Oveis Gharan, and Luca Trevisan. Multiway Spectral Partitioning and Higher-Order Cheeger Inequalities. *Journal of the ACM*, 61(6):37:1–37:30, 2014.
- 21 Richard Peng, He Sun, and Luca Zanetti. Partitioning Well-Clustered Graphs: Spectral Clustering Works! *SIAM Journal on Computing*, 46(2):710–743, 2017.
- 22 Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.
- 23 Amit Singer. Angular synchronization by eigenvectors and semidefinite programming. *Applied and computational harmonic analysis*, 30(1):20, 2011.
- 24 Daniel A. Spielman and Nikhil Srivastava. Graph Sparsification by Effective Resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.
- 25 David Steurer. Fast SDP Algorithms for Constraint Satisfaction Problems. In *25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA’10)*, pages 684–697, 2010.
- 26 Luca Trevisan. Approximation algorithms for unique games. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS’05)*, pages 197–205, 2005.
- 27 Luca Trevisan. Max Cut and the Smallest Eigenvalue. *SIAM Journal on Computing*, 41(6):1769–1786, 2012.

Packing Directed Circuits Quarter-Integrally

Tomáš Masařík

Department of Applied Mathematics, Charles University, Prague, Czech Republic & Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Warsaw, Poland
masarik@kam.mff.cuni.cz

Irene Muzi

Technische Universität Berlin, Germany
irene.muzi@tu-berlin.de

Marcin Pilipczuk

Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Warsaw, Poland
malcin@mimuw.edu.pl

Paweł Rzażewski

Faculty of Mathematics and Information Science, Warsaw University of Technology, Warsaw, Poland
p.rzazewski@mini.pw.edu.pl

Manuel Sorge

Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Warsaw, Poland
manuel.sorge@mimuw.edu.pl

Abstract

The celebrated Erdős-Pósa theorem states that every undirected graph that does not admit a family of k vertex-disjoint cycles contains a feedback vertex set (a set of vertices hitting all cycles in the graph) of size $\mathcal{O}(k \log k)$. After being known for long as Younger's conjecture, a similar statement for directed graphs has been proven in 1996 by Reed, Robertson, Seymour, and Thomas. However, in their proof, the dependency of the size of the feedback vertex set on the size of vertex-disjoint cycle packing is not elementary.

We show that if we compare the size of a minimum feedback vertex set in a directed graph with *quarter-integral* cycle packing number, we obtain a polynomial bound. More precisely, we show that if in a directed graph G there is no family of k cycles such that every vertex of G is in at most *four* of the cycles, then there exists a feedback vertex set in G of size $\mathcal{O}(k^4)$. On the way there we prove a more general result about quarter-integral packing of subgraphs of high directed treewidth: for every pair of positive integers a and b , if a directed graph G has directed treewidth $\Omega(a^6 b^8 \log^2(ab))$, then one can find in G a family of a subgraphs, each of directed treewidth at least b , such that every vertex of G is in at most four subgraphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Directed graphs, graph algorithms, linkage, Erdős-Pósa property

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.72

Related Version A full version of the paper is available at <http://arxiv.org/abs/1907.02494>.

Funding This research is a part of projects that have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme Grant Agreements 648527 (IM) and 714704 (all authors).



Tomáš Masařík: Also supported by Charles University, student grant number SVV-2017-260452.



Acknowledgements We thank Stephan Kreutzer (TU Berlin) for interesting discussions on the topic and for pointing out Lemma 3.



© Tomáš Masařík, Irene Muzi, Marcin Pilipczuk, Paweł Rzażewski, and Manuel Sorge; licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 72; pp. 72:1–72:13



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The theory of graph minors, developed over the span of over 20 years by Robertson and Seymour, had a tremendous impact on the area of graph algorithms. Arguably, one of the cornerstone contributions is the notion of *treewidth* [19] and the deep understanding of obstacles to small treewidth, primarily in the form of the *excluded grid theorem* [5, 20, 21].

Very tight relations of treewidth and the size of the largest grid as a minor in sparse graph classes, such as planar graphs or graphs excluding a fixed graph as a minor, led to the rich and fruitful theory of bidimensionality [10]. In general graphs, fine understanding of the existence of well-behaved highly-connected structures (not necessarily grids) in graphs of high treewidth has been crucial to the development of efficient approximation algorithms for the DISJOINT PATHS problem [9].

In undirected graphs, one of the first theorems that gave some well-behaved structure in a graph that is in some sense highly connected is the famous Erdős-Pósa theorem [11] linking the feedback vertex set number of a graph (the minimum number of vertices one needs to delete to obtain an acyclic graph) and the cycle packing number (the maximum possible size of a family of vertex-disjoint cycles in a graph). The Erdős-Pósa theorem states that a graph that does not contain a family of k vertex-disjoint cycles has feedback vertex set number bounded by $\mathcal{O}(k \log k)$.

A similar statement for directed graphs, asserting that a directed graph without a family of k vertex-disjoint cycles has feedback vertex set number at most $f(k)$, has been long known as the Younger's conjecture until finally proven by Reed, Robertson, Seymour, and Thomas in 1996 [17]. However, the function f obtained in [17] is not elementary; in particular, the proof relies on the Ramsey theorem for $\Theta(k)$ -regular hypergraphs. This is in contrast with the (tight) $\Theta(k \log k)$ bound in undirected graphs.

Our main result is that if one compares the feedback vertex set number of a directed graph to the *quarter-integral* cycle packing number (i.e., the maximum size of a family of cycles in G such that every vertex lies on at most four cycles), one obtains a polynomial bound.

► **Theorem 1.** *If a directed graph G does not contain a family of k cycles such that every vertex in G is contained in at most four cycles, then there exists a feedback vertex set in G of size $\mathcal{O}(k^4)$.*

We remark that if one relaxes the condition even further to a *fractional cycle packing*,¹ Seymour [22] proved that a graph without a fractional cycle packing of size at least k admits a feedback vertex set of size $\mathcal{O}(k \log k \log \log k)$.

Directed treewidth is a directed analog of the successful notion of treewidth, introduced in [13, 16]. An analog of the excluded grid theorem for directed graphs has been conjectured by Johnson, Robertson, Seymour, and Thomas [13] in 2001 and finally proven by Kawarabayashi and Kreutzer in 2015 [15]. Similarly as in the case of the directed Erdős-Pósa property, the relation between the directed treewidth of a graph and a largest directed grid as a minor in [15] is not elementary.

For a directed graph G , let $\text{fvs}(G)$, $\text{dtw}(G)$, and $\text{cp}(G)$ denote the feedback vertex set number, directed treewidth, and the cycle packing number of G , respectively. The following lemma is a restatement of the result of Amiri, Kawarabayashi, Kreutzer, and Wollan [1, Lemma 4.2]:

¹ A *fractional cycle packing* assigns to every cycle C in G a non-negative real weight $w(C)$ such that for every $v \in V(G)$ the total weight of all cycles containing v is at most 1. The *size* of a fractional cycle packing w is the total weight of all cycles in the packing.

► **Lemma 2** ([1, Lemma 4.2]). *Let G be a directed graph with $\text{dtw}(G) \leq w$. For each strongly connected directed graph H , the graph G has either k disjoint copies of H as a topological minor, or contains a set T of at most $k \cdot (w + 1)$ vertices such that H is not a topological minor of $G - T$.*

Note that the authors of [1] prove Lemma 2 for topological and butterfly minors, but the previous restatement is sufficient for our purposes. By taking H as the directed 2-cycle it is easy to derive the following bound:

► **Lemma 3.** *For a directed graph G it holds that $\text{fvs}(G) \leq (\text{dtw}(G) + 1) \text{cp}(G)$.*

In the light of Lemma 3 and since a directed grid minor of size k contains k vertex-disjoint cycles, the directed grid theorem of Kawarabayashi and Kreutzer [15] is a generalization of the directed Erdős-Pósa property due to Reed, Robertson, Seymour, and Thomas [17].

Theorem 1 is a direct corollary of Lemma 3 and the following statement that we prove.

► **Theorem 4.** *If a directed graph G does not contain a family of k cycles such that every vertex in G is contained in at most four cycles, then $\text{dtw}(G) = \mathcal{O}(k^3)$.*

Furthermore, if one asks not for a cycle packing, but a packing of subgraphs of large directed treewidth, we prove the following packing result.

► **Theorem 5.** *There exists an absolute constant c with the following property. For every pair of positive integers a and b , and every directed graph G of directed treewidth at least $c \cdot a^6 \cdot b^8 \cdot \log^2(ab)$, there are directed graphs G_1, G_2, \dots, G_a with the following properties:*

1. *each G_i is a subgraph of G ,*
2. *each vertex of G belongs to at most four graphs G_i , and*
3. *each graph G_i has directed treewidth at least b .*

Note that by setting $b = 2$ in Theorem 5, one obtains Theorem 4 with a slightly weaker bound of $\mathcal{O}(k^6 \log^2 k)$ and, consequently, Theorem 1 with a weaker bound of $\mathcal{O}(k^7 \log^2 k)$.

Theorem 5 should be compared to its undirected analog of Chekuri and Chuzhoy [4] that asserts that in an undirected graph G of treewidth at least $c \min(ab^2, a^3b)$ one can find a vertex-disjoint subgraphs of treewidth at least b . While we still obtain a polynomial bound, we can only prove the existence of a quarter-integral (as opposed to integral, i.e., vertex-disjoint) packing of subgraphs of high directed treewidth.

In the DISJOINT PATHS problem, given a graph G and a set of terminal pairs $(s_i, t_i)_{i=1}^k$, we ask to find an as large as possible collection of vertex-disjoint paths such that every path in the collection connects some s_i with t_i . Let OPT be the number of paths in the optimum solution; we say that a family \mathcal{P} is a *congestion- c polylogarithmic approximation* if every path in \mathcal{P} connects a distinct pair (s_i, t_i) , each vertex of $V(G)$ is contained in at most c paths of \mathcal{P} , and $|\mathcal{P}| \geq \text{OPT}/\text{polylog}(\text{OPT})$. The successful line of research of approximation algorithms for the DISJOINT PATHS problem in undirected graphs leading in particular to a congestion-2 polylogarithmic approximation algorithm of Chuzhoy and Li [9] for the edge-disjoint version, would not be possible without a fine understanding of well-behaved well-connected structures in a graph of high treewidth. Of central importance to such *routing* algorithms is the notion of a *crossbar*: a crossbar of order k and congestion c is a subgraph C of G with an *interface* $I \subseteq V(C)$ of size k such that for every matching M on I , one can connect the endpoints of the matching edges with paths in C such that every vertex is in at most c paths. Most of the known approximation algorithms for DISJOINT PATHS find a crossbar (C, I) with a large set of disjoint paths between I and the set of terminals s_i and t_i . While one usually does not control how the paths connect the terminals s_i and t_i to interface vertices of I , the ability of the crossbar to connect *any* given matching on the interface leads to a solution.

To obtain a polylogarithmic approximation algorithm, one needs the order of the crossbar to be comparable to the number of terminal pairs, which – by well-known tools such as *well-linked decompositions* [8] – is of the order of treewidth of the graph. At the same time, we usually allow constant congestion (every vertex can appear in a constant number of paths of the solution, instead of just one). Thus, the milestone graph-theoretic result used in approximation algorithms for DISJOINT PATHS is the existence of a congestion-2 crossbar of order k in a graph of treewidth $\Omega(k \text{polylog}(k))$.

While the existence of similar results for the general DISJOINT PATHS problem in directed graphs is implausible [2], Chekuri, and Ene proposed to study the case of *symmetric demands* where one asks for a path from s_i to t_i and a path from t_i to s_i for a terminal pair (s_i, t_i) . First, they provided an analog of the well-linked decomposition for this case [6], and then with Pilipczuk [7] showed an existence of an analog of a crossbar and a resulting approximation algorithm for DISJOINT PATHS with symmetric demands in planar directed graphs. Later, this result has been lifted to arbitrary proper minor-closed graph classes [3]. However, the general case remains widely open.

As discussed above, for applications in approximation algorithms for DISJOINT PATHS, it is absolutely essential to squeeze as much as possible from the bound linking directed treewidth of a graph with the order of the crossbar, while the final congestion is of secondary importance (but we would like it to be a small constant). We think of Theorem 5 as a step in this direction: sacrificing integral packings for quarter-integral ones, we obtain much stronger bounds than the non-elementary bounds of [17]. Furthermore, such a step seems necessary, as it is hard to imagine a crossbar of order k that would not contain a constant-congestion (i.e., every vertex used in a constant number of cycles) packing of $\Omega(k)$ directed cycles.

On the technical side, the proof of Theorem 5 borrows a number of technical tools from the recent work of Hatzel, Kawarabayashi, and Kreuzer that proved polynomial bounds for the directed grid minor theorem in planar graphs [12]. We follow their general approach to obtain a directed treewidth sparsifier [12, Section 5] and modify it in a number of places for our goal. The main novelty comes in different handling of the case when two linkages intersect a lot. Here we introduce a new partitioning tool (see Section 3) which we use in the crucial moment where we separate subgraphs G_i from each other.

Organization. After brief preliminaries in Section 2, we prove Theorem 5 in Sections 3–5: Section 3 introduces the new partitioning tool, Section 4 handles the most complicated “dense case” in the analysis, while Section 5 wraps up the argument. Discussions on the the adaptation of the arguments of Section 5 to obtain the improved bound of Theorem 4 and some argumentation from Section 4 that directly follows the arguments of [12] can be found in the full version of the paper.

2 Preliminaries

Let $G = (V(G), E(G))$ be a directed graph and let A, B be subsets of $V(G)$ with $|A| = |B|$. A *linkage* from A to B in G is a set \mathcal{L} of $|A|$ pairwise vertex-disjoint paths in G , each with a starting vertex in A and ending vertex in B . The *order* of \mathcal{L} is $|\mathcal{L}| = |A|$. For $X, Y \subseteq V(G)$ and a linkage \mathcal{L} from X to Y , we denote $A(\mathcal{L}) := X$ and $B(\mathcal{L}) := Y$. For a path or a walk P , by $\text{start}(P)$ and $\text{end}(P)$ we denote the starting and ending vertex of P , respectively.

Let \mathcal{L} and \mathcal{K} be linkages. The *intersection graph* of \mathcal{L} and \mathcal{K} , denoted by $I(\mathcal{L}, \mathcal{K})$, is the bipartite graph with the vertex set $\mathcal{L} \cup \mathcal{K}$ and an edge between a vertex in \mathcal{L} and a vertex in \mathcal{K} if the corresponding paths share at least one vertex.

A vertex set $W \subseteq V(G)$ is *well-linked* if for all subsets $A, B \subseteq W$ with $|A| = |B|$ there is a linkage \mathcal{L} of order $|A|$ from A to B in $G \setminus (W \setminus (A \cup B))$.

Let \mathcal{P} be a family of walks in G and let c be a positive integer. We say that \mathcal{P} is of *congestion* c if for every $v \in V(G)$, the total number of times the walks in \mathcal{P} visit v is at most c ; here, if a walk $W \in \mathcal{P}$ visits v multiple times, we count each visit separately. A family of paths \mathcal{P} is a *half-integral* (*quarter-integral*) if it is of congestion 2 (resp. 4).

We call two linkages \mathcal{L} and $\mathcal{L}^{\text{back}}$ *dual* to each other if $A(\mathcal{L}) = B(\mathcal{L}^{\text{back}})$ and $A(\mathcal{L}^{\text{back}}) = B(\mathcal{L})$. For two dual linkages \mathcal{L} and $\mathcal{L}^{\text{back}}$ in a graph G , we define an auxiliary directed graph $\text{Aux}(\mathcal{L}, \mathcal{L}^{\text{back}})$ as follows. We take $V(\text{Aux}(\mathcal{L}, \mathcal{L}^{\text{back}})) = \mathcal{L}$ and for every path $P \in \mathcal{L}^{\text{back}}$ that starts in a vertex $\text{start}(P) = \text{end}(L)$ for some $L \in \mathcal{L}$ and ends in a vertex $\text{end}(P) = \text{start}(L')$ for some $L' \in \mathcal{L}$, we put an arc (L, L') to $\text{Aux}(\mathcal{L}, \mathcal{L}^{\text{back}})$. Note that it may happen that $L = L'$. When the backlinkage $\mathcal{L}^{\text{back}}$ is clear from the context, we abbreviate $\text{Aux}(\mathcal{L}, \mathcal{L}^{\text{back}})$ to $\text{Aux}(\mathcal{L})$. Observe that in $\text{Aux}(\mathcal{L}, \mathcal{L}^{\text{back}})$ every node is of in- and out-degree exactly one and thus this graph is a disjoint union of directed cycles.

With every arc (L, L') of $\text{Aux}(\mathcal{L}, \mathcal{L}^{\text{back}})$ we can associate the walk from $\text{start}(L)$ to $\text{start}(L')$ that first goes along L and then follows the path $P \in \mathcal{L}^{\text{back}}$ that gives rise to the arc (L, L') . Consequently, with every collection of pairwise disjoint paths and cycles in $\text{Aux}(\mathcal{L}, \mathcal{L}^{\text{back}})$ there is an associated collection of walks (closed walks for cycles) in G that is of congestion 2 as it originated from two linkages. Note that the same construction works if \mathcal{L} and $\mathcal{L}^{\text{back}}$ are half-integral linkages, and then the walks in G corresponding to a family of paths and cycles in $\text{Aux}(\mathcal{L}, \mathcal{L}^{\text{back}})$ would be of congestion 4.

Furthermore, with a pair of dual linkages \mathcal{L} and $\mathcal{L}^{\text{back}}$ we can associate a *backlinkage-induced order* $\mathcal{L} = \{L_1, L_2, \dots, L_{|\mathcal{L}|}\}$ as follows. If C_1, C_2, \dots, C_r are the cycles of the graph $\text{Aux}(\mathcal{L}, \mathcal{L}^{\text{back}})$ in an arbitrary order, then $L_1, L_2, \dots, L_{|C_1|}$ are the vertices of C_1 in the order of their appearance on C_1 , and $L_{|C_1|+1}, \dots, L_{|C_1|+|C_2|}$ are the vertices of C_2 in the order of their appearance on C_2 , etc. That is, we order the elements of \mathcal{L} first according to the cycle of $\text{Aux}(\mathcal{L})$ they lie on, and then, within one cycle, according to the order around this cycle.

We will also need the following operation on a pair of dual linkages \mathcal{L} and $\mathcal{L}^{\text{back}}$. Let $\mathcal{P} \subseteq \mathcal{L}$ be a sublinkage. For every $P \in \mathcal{P}$, construct a walk $Q(P)$ as follows. Start from the path $Q_0 \in \mathcal{L}^{\text{back}}$ with $\text{start}(Q_0) = \text{end}(P)$ and set $Q(P) = Q_0$. Given $Q_i \in \mathcal{L}^{\text{back}}$ for $i \geq 0$, proceed as follows. Let $P_{i+1} \in \mathcal{L}$ be the path with $\text{end}(Q_i) = \text{start}(P_{i+1})$. If $P_{i+1} \in \mathcal{P}$, then stop. Otherwise, define $Q_{i+1} \in \mathcal{L}^{\text{back}}$ to be the path with $\text{end}(P_{i+1}) = \text{start}(Q_{i+1})$. Append P_{i+1} and Q_{i+1} at the end of $Q(P)$ and repeat. Finally, we shortcut $Q(P)$ to a path $Q'(P)$ with the same endpoints. In this manner, $\mathcal{Q} := \{Q'(P) \mid P \in \mathcal{P}\}$ is a half-integral linkage with $A(\mathcal{P}) = B(\mathcal{Q})$ and $A(\mathcal{Q}) = B(\mathcal{P})$. We call \mathcal{Q} the *backlinkage induced by \mathcal{P} on $(\mathcal{L}, \mathcal{L}^{\text{back}})$* . Furthermore, we can perform the same construction if \mathcal{L} and $\mathcal{L}^{\text{back}}$ are half-integral linkages, obtaining a quarter-integral linkage \mathcal{Q} .

We say that G is d -degenerate if and only if every subgraph of G contains a vertex of degree at most d .

In this paper we do not need the exact definition of directed treewidth. Instead, we rely on the following two results.

► **Lemma 6** ([16]). *Every directed graph G of directed treewidth k contains a well-linked set of size $\Omega(k)$.*

► **Lemma 7** ([14, 15]). *There is an absolute constant c' with the following property. Let $\alpha, \beta \geq 1$ be integers and let G be a digraph of $\text{dtw}(G) \geq c' \cdot \alpha^2 \beta^2$. Then there exists a set of α vertex-disjoint paths P_1, \dots, P_α and sets $A_i, B_i \subseteq V(P_i)$, where A_i appears before B_i on P_i , both $|A_i|, |B_i| = \beta$, and $\bigcup_{i=1}^\alpha A_i \cup B_i$ is well-linked.*

We also need the following two auxiliary results. Note that a coloring in Lemma 8 can be arbitrary and is not necessarily proper.

► **Lemma 8** ([18, Lemma 4.3]). *Let $r \geq 2$, d be a real, and H be an r -colored graph with color classes V_1, \dots, V_r , such that for every i it holds that $|V_i| \geq 4e(r-1)d$ and for every $i \neq j$ the graph $H[V_i \cup V_j]$ is d -degenerate. Then there exists an independent set $\{x_1, \dots, x_r\}$ such that $x_i \in V_i$ for every $i \in [r]$.*

► **Lemma 9** ([12, Lemma 5.5]). *Let G be a digraph and P_1, \dots, P_k be disjoint paths such that each P_i consists of two subpaths A_i and B_i , where A_i precedes B_i . Furthermore, let $\{L_{i,j} : i, j \in [k], i \neq j\}$ be a set of pairwise disjoint paths, such that $L_{i,j}$ starts in B_i and ends in A_j . Then*

$$\text{dtw}\left(\bigcup_i P_i \cup \bigcup_{i \neq j} L_{i,j}\right) \geq \frac{k}{8}.$$

3 Partitioning Lemma

In this section, we develop a main technical tool that we use in the proof of Theorem 5. Intuitively, in a subcase of the proof, we will have a bipartite graph of large minimum degree which we partition into subgraphs induced by pairs of vertex sets (U_i, W_i) . These subgraphs will define the G_i from the statement of Theorem 5. To obtain a lower bound on the directed treewidth of G_i , we need that the parts (U_i, W_i) each induce a subgraph of large average degree. This will be achieved using the following lemma.

► **Lemma 10.** *Let $h \geq 0$ and n be integers, d be a positive real such that $d \cdot 4^{h+1} - 1 > 2$, and let G be a bipartite graph with bipartition classes $X = \{x_1, x_2, \dots, x_a\}$ and $Y = \{y_1, y_2, \dots, y_b\}$, such that $a + b \leq n$ and $|E(G)| \geq (d \cdot 4^{h+1} - 1) \cdot n$. Then in $[a]$ we can find $k := 2^h$ pairwise disjoint sets I_1, I_2, \dots, I_k , and in $[b]$ we can find k pairwise disjoint sets J_1, J_2, \dots, J_k , such that:*

1. for every $i \in [k]$ the set I_i is a segment of $[a]$ and the set J_i is a segment of $[b]$,
2. for every $i \in [k]$, the number of edges between $\{x_i : i \in I_i\}$ and $\{y_i : i \in J_i\}$ is at least $d \cdot n$.

Proof. For $I \subseteq [a]$ and $J \subseteq [b]$, let $e(I, J)$ we denote the number of edges $x_i y_j$ of G , such that $i \in I$ and $j \in J$. Observe that $|E(G)| > 2n$.

We prove the lemma by induction on h . Note that for $h = 0$ the claim is trivially satisfied by taking $I_1 = X$ and $J_1 = Y$, as $d \cdot 4^{h+1} - 1 > 2$ and $h \geq 0$ implies $d \cdot 4^{h+1} - 1 \geq d$. So now assume that $h \geq 1$ and the claim holds for $h - 1$. Let $s \in [a]$ be the minimum integer, for which $\sum_{i=1}^s \deg x_i \geq |E(G)|/2$, and let $t \in [b]$ be the minimum integer, for which $\sum_{i=1}^t \deg y_i \geq |E(G)|/2$. We observe that $d \cdot 4^{h+1} - 1 > 2$ implies that $1 < s < a$ and $1 < t < b$. Define $X^1 := \{1, 2, \dots, s-1\}$ and $X^2 := \{s+1, \dots, a\}$, and $Y^1 := \{1, 2, \dots, t-1\}$ and $Y^2 := \{t+1, \dots, b\}$.

We aim to show that the number of edges joining X^1 and Y^1 is roughly the same as the number of edges joining X^2 and Y^2 , and the number of edges joining X^1 and Y^2 is roughly the same as the number of edges joining X^2 and Y^1 . Since $\deg x_s \leq b < n$ and $\deg y_t \leq a < n$, by the choice of s and t we obtain the following set of inequalities.

$$\begin{aligned}
 e(X, Y)/2 - \deg x_s &\leq e(X^1, Y) \leq e(X, Y)/2 \\
 e(X, Y)/2 - \deg x_s &\leq e(X^2, Y) \leq e(X, Y)/2 \\
 e(X, Y)/2 - \deg y_t &\leq e(X, Y^1) \leq e(X, Y)/2 \\
 e(X, Y)/2 - \deg y_t &\leq e(X, Y^2) \leq e(X, Y)/2.
 \end{aligned} \tag{1}$$

Observe that

$$\begin{aligned}
 e(X^1, Y^1) + e(X^1, Y^2) &\leq e(X^1, Y) = e(X^1, Y^1) + e(X^1, Y^2) + e(X^1, \{t\}) \\
 &\leq e(X^1, Y^1) + e(X^1, Y^2) + \deg y_t
 \end{aligned}$$

(and analogously for each of the remaining inequalities in (1)). Thus we obtain:

$$\begin{aligned}
 e(X, Y)/2 - n &\leq e(X^1, Y^1) + e(X^1, Y^2) \leq e(X, Y)/2 \\
 e(X, Y)/2 - n &\leq e(X^2, Y^1) + e(X^2, Y^2) \leq e(X, Y)/2 \\
 e(X, Y)/2 - n &\leq e(X^1, Y^2) + e(X^2, Y^1) \leq e(X, Y)/2 \\
 e(X, Y)/2 - n &\leq e(X^1, Y^2) + e(X^2, Y^2) \leq e(X, Y)/2.
 \end{aligned} \tag{2}$$

By subtracting appropriate pairs of inequalities in (2), we obtain the following bounds.

$$\begin{aligned}
 -n &\leq e(X^1, Y^1) - e(X^2, Y^2) \leq n \\
 -n &\leq e(X^1, Y^2) - e(X^2, Y^1) \leq n
 \end{aligned} \tag{3}$$

Recall that

$$\begin{aligned}
 e(X, Y) &= e(X^1, Y^1) + e(X^1, Y^2) + e(X^2, Y^1) + e(X^2, Y^2) + \deg x_s + \deg y_t \\
 &\leq e(X^1, Y^1) + e(X^1, Y^2) + e(X^2, Y^1) + e(X^2, Y^2) + n.
 \end{aligned}$$

Thus, by the pigeonhole principle, at least one of the following holds:

$$\begin{aligned}
 e(X^1, Y^1) + e(X^2, Y^2) &\geq e(X, Y)/2 - n/2 \\
 e(X^1, Y^2) + e(X^2, Y^1) &\geq e(X, Y)/2 - n/2.
 \end{aligned} \tag{4}$$

Suppose that the first case holds. Define $G^1 = G[X^1 \cup Y^1]$ and $G^2 = G[X^2 \cup Y^2]$. Combining (3) and (4), we obtain that

$$\begin{aligned}
 |E(G^1)| = e(X^1, Y^1) &\geq e(X, Y)/4 - 3n/4 \geq (d \cdot 4^{h+1} - 1)n/4 - 3n/4 = (d \cdot 4^h - 1)n \\
 |E(G^2)| = e(X^2, Y^2) &\geq e(X, Y)/4 - 3n/4 \geq (d \cdot 4^h - 1)n.
 \end{aligned} \tag{5}$$

We observe that graphs G^1, G^2 satisfy the inductive assumption (for $h - 1$), so in the vertex set of G^1 we can find two families of $k/2$ pairwise corresponding segments $I_1^1, I_2^1, \dots, I_{k/2}^1$ and $J_1^1, J_2^1, \dots, J_{k/2}^1$, and in the vertex set of G^2 we can find two families of $k/2$ pairwise corresponding segments $I_1^2, I_2^2, \dots, I_{k/2}^2$ and $J_1^2, J_2^2, \dots, J_{k/2}^2$. We obtain the desired subsegments of X and Y by setting:

$$I_i = \begin{cases} I_i^1 & \text{if } i \leq k/2, \\ I_{i-k/2}^2 & \text{if } i > k/2, \end{cases} \quad J_i = \begin{cases} J_i^1 & \text{if } i \leq k/2, \\ J_{i-k/2}^2 & \text{if } i > k/2. \end{cases}$$

If the second case in (4) holds, we take $G^1 = G[X^1, Y^2]$ and $G^2 = G[X^2, Y^1]$, and the rest of the proof is analogous. \blacktriangleleft

72:8 Packing Directed Circuits Quarter-Integrally

The following statement brings the technical statement of Lemma 10 into a more easily applicable form.

► **Lemma 11.** *Let $k, r \geq 1$ be two integers and let G be a bipartite graph with bipartition classes $X = \{x_1, x_2, \dots, x_a\}$ and $Y = \{y_1, y_2, \dots, y_b\}$ and minimum degree at least $1200 \cdot r \cdot k$. Then there are k sets U_1, U_2, \dots, U_k , and k sets W_1, W_2, \dots, W_k , such that:*

1. *for each $i \in [k]$ the set U_i is a segment of $[a]$ and the set W_i is a segment of $[b]$,*
2. *for each distinct $i, j \in [k]$ we have $U_i \cap U_j = \emptyset$ and $W_i \cap W_j = \emptyset$,*
3. *for every $i \in [k]$, the average degree of the graph $G[U_i \cup W_i]$ is at least r .*

Proof. Let h be the minimum integer, such that $k' := 2^h \geq 3k$; note that $k' < 6k$. Also, define $d = 2r/k$ and $n = a + b$. We have

$$d \cdot 4^{h+1} - 1 = 4d(k')^2 - 1 \geq \frac{8r}{k} \cdot (3k)^2 - 1 = 72 \cdot r \cdot k - 1 > 2.$$

Observe that the number of edges in G is at least

$$n \cdot r \cdot 600k > (16r/k \cdot (6k)^2)n > (4d(k')^2)n > (d \cdot 4^{h+1} - 1)n.$$

Thus G satisfies the assumptions of Lemma 10 for h , n , and d . Let $I_1, I_2, \dots, I_{k'}$ be the disjoint segments in X , and $J_1, J_2, \dots, J_{k'}$ be the disjoint segments in Y , whose existence is guaranteed by Lemma 10.

A segment I_i (J_i , resp.) is called *large* if $|I_i| \geq 3n/k'$ ($|J_i| \geq 3n/k'$, resp.). A pair (I_i, J_i) is *large* if at least one of I_i, J_i is large, otherwise the pair is *small*. Note that there are at most $n/(3n/k') = k'/3$ large segments I_i and at most $k'/3$ large segments J_i , so the number of large pairs is at most $2k'/3$. Thus the number of small pairs is at least $k'/3 \geq k$. We obtain the segments (U_i, W_i) by taking the first k small pairs (I_i, J_i) . Clearly these segments satisfy conditions 1. and 2. of the lemma.

Now take any $i \in [k]$ and let us compute the average degree of the graph $G_i := G[U_i, W_i]$. By Lemma 10, $|E(G_i)| \geq d \cdot n$. On the other hand, since (U_i, W_i) is a small pair, we have that $|V(G_i)| = |U_i \cup W_i| < 6n/k'$. Thus we obtain that the average degree of G_i is

$$\frac{|E(G_i)|}{|V(G_i)|} > \frac{d \cdot n}{6n/k'} = \frac{dk'}{6} \geq d \frac{3k}{6} = \frac{2r}{k} \cdot \frac{k}{2} = r.$$

This completes the proof. ◀

4 The Dense Case

In this section, we prove Theorem 5 roughly in the case when there are two linkages \mathcal{L} and \mathcal{K} such that their set $A(\mathcal{L}) \cup A(\mathcal{K}) \cup B(\mathcal{L}) \cup B(\mathcal{K})$ of endpoints is well linked and such that the paths in \mathcal{L} and \mathcal{K} intersect a lot. The formal statement proved in this section is as follows.

► **Lemma 12.** *Let $a, b \in \mathbb{N}^+$. Let D be a directed graph and \mathcal{L} and \mathcal{K} be two linkages in D such that $A(\mathcal{L}) \cup B(\mathcal{L}) \cup A(\mathcal{K}) \cup B(\mathcal{K})$ is well-linked in D . Suppose that the intersection graph $I(\mathcal{L}, \mathcal{K})$ has degeneracy more than $384000 \cdot a \cdot b \cdot \log_2(|\mathcal{L}|/b)$. Then there are directed graphs D_1, D_2, \dots, D_a with the following properties:*

- (i) *each D_i is a subgraph of D ,*
- (ii) *each vertex of D belongs to at most four graphs D_i , and*
- (iii) *each graph D_i has directed treewidth at least b .*

Proof Outline. The basic idea of the proof of Lemma 12 is as follows. We first fix a pair of linkages $\mathcal{L}^{\text{back}}$ and $\mathcal{K}^{\text{back}}$ which are dual to \mathcal{L} and \mathcal{K} , respectively. (This is possible because of well-linkedness of the endpoints.) The subgraphs D_i that we construct will subpartition the vertex set of each of the four linkages $\mathcal{L}, \mathcal{L}^{\text{back}}, \mathcal{K}, \mathcal{K}^{\text{back}}$ and hence each vertex of G is in at most four subgraphs D_i . To construct the desired subgraphs D_i , we consider the backlinkage-induced order $\Pi_{\mathcal{L}}$ on \mathcal{L} and $\Pi_{\mathcal{K}}$ on \mathcal{K} . Using these orderings of the paths of \mathcal{L} and \mathcal{K} , we can apply the partitioning lemma (Lemma 11) to the intersection graph of \mathcal{L} and \mathcal{K} , obtaining a subpartition I_1, \dots, I_k of \mathcal{L} and a subpartition J_1, \dots, J_k of \mathcal{K} . These subpartitions have the nice property that each intersection graph $I(I_i, J_i)$ induced by a pair I_i, J_i contains many edges (representing intersections between the corresponding paths) and that only a constant number of cycles of $\text{Aux}(\mathcal{L})$ and $\text{Aux}(\mathcal{K})$ cross I_i or J_i . By closing each of these crossing cycles by introducing an artificial new path, we obtain a pair of dual linkages I_i, I'_i , and a pair of dual of linkages J_i, J'_i . Using then Lemma 13 below, we will obtain a lower bound on the directed treewidth of the graph induced by $I_i \cup J_i \cup I'_i \cup J'_i$, which constitute our desired subgraph D_i .

Treewidth Lower Bound. For technical reasons, we will have to work with half-integral linkages. The intersection graph for a pair of half-integral linkages is defined in the same way as for ordinary linkages.

► **Lemma 13.** *Let $k, d \in \mathbb{N}^+$ and $\mathcal{P}, \mathcal{P}^{\text{back}}, \mathcal{Q}, \mathcal{Q}^{\text{back}}$ be four half-integral linkages in a directed graph such that \mathcal{P} and $\mathcal{P}^{\text{back}}$ are dual to each other and \mathcal{Q} and $\mathcal{Q}^{\text{back}}$ are dual to each other. Let the intersection graph $I(\mathcal{P}, \mathcal{Q})$ have minimum degree at least d where $d \geq 8k \log_{\frac{4}{3}} \left(\frac{|\mathcal{P}|}{24k} \right) + 24k + 4$. Then the graph $\bigcup (\mathcal{P} \cup \mathcal{P}^{\text{back}} \cup \mathcal{Q} \cup \mathcal{Q}^{\text{back}})$ has directed treewidth at least k .*

The proof of Lemma 13 is inspired by the proof of Lemma 5.4 in [12]. We could use Lemma 5.4 here as well, but its proof, unfortunately, contains errors. Nevertheless, we derive an incomparable bound which is much better for our use since the lower bound claimed in Lemma 5.4 [12] is k^2 . Also, we adapt the constants in the lemma for half-integral linkages. We postpone the proof of Lemma 13 to the full version of the paper.

Main Proof of the Dense Case. We are now ready to prove the main lemma of this section.

Proof of Lemma 12. Let $d = 384\,000 \cdot a \cdot b \cdot \log_2(|\mathcal{L}|/b)$. Since $I(\mathcal{L}, \mathcal{K})$ is not d -degenerate, it contains an induced subgraph I' of minimum degree larger than d . Redefine \mathcal{L} and \mathcal{K} to be the sublinkages of \mathcal{L} and \mathcal{K} contained in this subgraph I' , that is, $\mathcal{L} := \mathcal{L} \cap V(I')$ and $\mathcal{K} := \mathcal{K} \cap V(I')$. Note that $|\mathcal{L}| > d$, $|\mathcal{K}| > d$, the size of \mathcal{L} only decreases, and that $A(\mathcal{L}) \cup B(\mathcal{L}) \cup A(\mathcal{K}) \cup B(\mathcal{K})$ remains well-linked.

Let $\mathcal{L}^{\text{back}}$ be a linkage in D from $B(\mathcal{L})$ to $A(\mathcal{L})$ and let $\mathcal{K}^{\text{back}}$ be a linkage in D from $B(\mathcal{K})$ to $A(\mathcal{K})$. Note that $\mathcal{L}^{\text{back}}$ and $\mathcal{K}^{\text{back}}$ exist because $A(\mathcal{L}) \cup B(\mathcal{L}) \cup A(\mathcal{K}) \cup B(\mathcal{K})$ is well linked.

We focus on $\text{Aux}(\mathcal{L})$ and $\text{Aux}(\mathcal{K})$. Take backlinkage-induced orderings $(L_1, \dots, L_{|\mathcal{L}|})$ of \mathcal{L} and $(K_1, \dots, K_{|\mathcal{K}|})$ of \mathcal{K} . Apply Lemma 11 with $k = a$, $r = 320b \log_2(|\mathcal{L}|/b)$, $G = I(\mathcal{L}, \mathcal{K})$, $X = \{L_1, \dots, L_{|\mathcal{L}|}\}$, and $Y = \{K_1, \dots, K_{|\mathcal{K}|}\}$, obtaining a sets U_1, \dots, U_a and a sets W_1, \dots, W_a with the corresponding properties. To see that Lemma 11 is applicable, observe that $I(\mathcal{L}, \mathcal{K})$ has minimum degree at least $384\,000 \cdot a \cdot b \log_2(|\mathcal{L}|/b) = 1200 \cdot 320b \log_2(|\mathcal{L}|/b) \cdot a = 1200 \cdot r \cdot k$. Observe for later on that, for each $i \in [a]$, the intersection graph $I(U_i, W_i)$ of the two linkages U_i and W_i has average degree at least $320b \log_2(|\mathcal{L}|/b)$ by property 3 of Lemma 11.

72:10 Packing Directed Circuits Quarter-Integrally

Now define, for each $i \in [a]$, a graph D_i as follows. Initially, take the union of all paths in U_i and W_i . Then, for each edge (L, L') of $\text{Aux}(\mathcal{L})$ such that $L, L' \in U_i$, add to D_i the unique path $P \in \mathcal{L}^{\text{back}}$ that connects L and L' , that is, $\text{end}(L) = \text{start}(P)$ and $\text{end}(P) = \text{start}(L')$. Similarly, for each edge (K, K') of $\text{Aux}(\mathcal{K})$ such that $K, K' \in W_i$, add to D_i the unique path $Q \in \mathcal{K}^{\text{back}}$ with $\text{end}(K) = \text{start}(Q)$ and $\text{end}(Q) = \text{start}(K')$. In formulas:

$$U'_i := \{P \in \mathcal{L}^{\text{back}} \mid \exists(L, L') \in E(\text{Aux}(\mathcal{L})):$$

$$L, L' \in U_i \wedge \text{end}(L) = \text{start}(P) \wedge \text{end}(P) = \text{start}(L')\}$$

and

$$W'_i := \{Q \in \mathcal{K}^{\text{back}} \mid \exists(K, K') \in E(\text{Aux}(\mathcal{K})):$$

$$K, K' \in W_i \wedge \text{end}(K) = \text{start}(Q) \wedge \text{end}(Q) = \text{start}(K')\}.$$

We set

$$D_i := \bigcup(U_i \cup W_i \cup U'_i \cup W'_i).$$

We claim that D_i satisfies the required properties. Clearly, D_i is a subgraph of D , giving property (i). To see property (ii), consider a linkage $\mathcal{P} \in \{\mathcal{L}, \mathcal{L}^{\text{back}}, \mathcal{K}, \mathcal{K}^{\text{back}}\}$. We claim that no two subgraphs D_i, D_j contain the same path of \mathcal{P} . This claim follows indeed from property 2. of Lemma 11, stating that $U_i \cap U_j = \emptyset$ and $W_i \cap W_j = \emptyset$ and inspecting the definition of D_i and D_j . Thus, $\{V(D_i) \mid i \in [a]\}$ is a partition of a subset of the vertex set $V(\mathcal{P})$ of the paths in \mathcal{P} . Thus, each vertex $v \in V(D)$ occurs in at most four subgraphs D_i , showing property (ii).

It remains to show property (iii), the lower bound on the directed treewidth of D_i . We aim to modify D_i , increasing the directed treewidth by at most a constant, to obtain a graph $D_i^{(2)}$ which is the union of two pairs of dual half-integral linkages such that two linkages contained in distinct pairs intersect a lot. Then we can apply Lemma 13, giving a lower bound on the directed treewidth of $D_i^{(2)}$ which then implies a lower bound on the directed treewidth of D_i .

We first modify D_i to obtain a graph $D_i^{(1)}$ which is the union of two pairs of dual linkages. Recall the orderings $\vec{\mathcal{L}} := (L_1, \dots, L_{|\mathcal{L}|})$ and $\vec{\mathcal{K}} := (K_1, \dots, K_{|\mathcal{K}|})$ on \mathcal{L} and \mathcal{K} , respectively, which we have defined above. By property 1. of Lemma 11, U_i is a segment of $\vec{\mathcal{L}}$ and W_i is a segment of $\vec{\mathcal{K}}$. Hence, by the way we have defined $\vec{\mathcal{L}}$, there are at most two cycles C in $\text{Aux}(\mathcal{L})$ which are not contained in U_i or disjoint with U_i , that is $V(C) \setminus U_i \neq \emptyset$ and $V(C) \cap U_i \neq \emptyset$. Call such a cycle *broken*. Similarly, there are at most two cycles C in $\text{Aux}(\mathcal{K})$ such that $V(C) \setminus W_i \neq \emptyset$ and $V(C) \cap W_i \neq \emptyset$. Call such a cycle *broken* as well. For each broken cycle C , do the following operation on D_i to obtain $D_i^{(1)}$. If C is in $\text{Aux}(\mathcal{L})$, let L_{out}^C be the vertex of outdegree zero in the subgraph $\text{Aux}(\mathcal{L})[V(C) \cap U_i]$ and let L_{in}^C be the vertex of indegree zero. Add the directed edge $(\text{end}(L_{\text{out}}^C), \text{start}(L_{\text{in}}^C))$ to D_i . Proceed analogously if C is in $\text{Aux}(\mathcal{K})$: Let K_{out}^C be the vertex of outdegree zero in the subgraph $\text{Aux}(\mathcal{K})[V(C) \cap W_i]$ and let K_{in}^C be the vertex of indegree zero, and add the directed edge $(\text{end}(K_{\text{out}}^C), \text{start}(K_{\text{in}}^C))$ to D_i . In this way, we add at most four edges to D_i , obtaining $D_i^{(1)}$. Note that adding an edge increases the directed treewidth by at most one², and hence $\text{dtw}(D_i^{(1)}) \leq \text{dtw}(D_i) + 4$.

² In the corresponding robber-cop game (see [13]), we can always guard the new edge with an additional cop.

We claim that $D_i^{(1)}$ is the union of two pairs of dual linkages. To see this, note first that U_i and W_i are linkages in $D_i^{(1)}$. Now consider

$$U_i^b := U_i' \cup \{(\text{end}(L_{\text{out}}^C), \text{start}(L_{\text{in}}^C)) \mid C \text{ a broken cycle in } \text{Aux}(\mathcal{L})\}$$

and

$$W_i^b := W_i' \cup \{(\text{end}(K_{\text{out}}^C), \text{start}(K_{\text{in}}^C)) \mid C \text{ a broken cycle in } \text{Aux}(\mathcal{K})\},$$

wherein $L_{\text{in}}^C, L_{\text{out}}^C, K_{\text{in}}^C$, and K_{out}^C are defined as above. Clearly, $D_i^{(1)} = \bigcup(U_i \cup W_i \cup U_i^b \cup W_i^b)$. Moreover, both U_i^b and W_i^b are linkages because U_i' and W_i' are linkages and because $L_{\text{in}}^C, L_{\text{out}}^C, K_{\text{in}}^C$, and K_{out}^C have indegree or outdegree zero in $\text{Aux}(\mathcal{L})[V(C)]$ or $\text{Aux}(\mathcal{K})[V(C)]$, respectively. Finally, by definition, U_i and U_i^b are dual to each other and W_i and W_i^b are dual to each other. Thus, $D_i^{(1)}$ is the union of two pairs of dual linkages, as claimed.

In order to apply Lemma 13, we need a pair of linkages whose intersection graph has a large minimum degree. So far, the linkages which define $D_i^{(1)}$ guarantee only large average degree (via property 3. of Lemma 11). We now derive a subgraph $D_i^{(2)}$ of $D_i^{(1)}$ such that $D_i^{(2)}$ is the union of two pairs of dual half-integral linkages $(\mathcal{P}, \mathcal{P}^{\text{back}}), (\mathcal{Q}, \mathcal{Q}^{\text{back}})$ and $I(\mathcal{P}, \mathcal{Q})$ has large minimum degree. To achieve this, recall that the intersection graph $I(U_i, W_i)$ of the two linkages U_i, W_i in $D_i^{(1)}$ has average degree at least $320b \log_2(|\mathcal{L}|/b)$. Hence, there is a subgraph I' of $I(U_i, W_i)$ with minimum degree at least $320b \log_2(|\mathcal{L}|/b)$. Let $\mathcal{P} \subseteq U_i$ be the sublinkage of U_i contained in I' , that is $\mathcal{P} = U_i \cap V(I')$. Similarly, let $\mathcal{Q} = W_i \cap V(I')$.

We define $\mathcal{P}^{\text{back}}$ to be the backlinkage induced by \mathcal{P} on (U_i, U_i^b) and $\mathcal{Q}^{\text{back}}$ to be the backlinkage induced by \mathcal{Q} on (W_i, W_i^b) . Note that $\mathcal{P}^{\text{back}}$ and $\mathcal{Q}^{\text{back}}$ are half-integral and dual to \mathcal{P} and \mathcal{Q} , respectively.

Take now the subgraph $D_i^{(2)}$ to be the union $\bigcup(\mathcal{P} \cup \mathcal{P}^{\text{back}} \cup \mathcal{Q} \cup \mathcal{Q}^{\text{back}})$. Then apply Lemma 13 to $\mathcal{P}, \mathcal{P}^{\text{back}}, \mathcal{Q}, \mathcal{Q}^{\text{back}}$ with $k = b + 4$ and $d = 320b \log_2(|\mathcal{L}|/b)$. To see that the preconditions of Lemma 13 are satisfied, first recall that the intersection graph $I(\mathcal{P}, \mathcal{Q})$ has minimum degree at least $320b \log_2(|\mathcal{L}|/b)$. Furthermore,

$$\begin{aligned} d &= 320b \log_2 \frac{|\mathcal{L}|}{b} \geq 200b \log_2 \frac{|\mathcal{L}|}{b} + 120b + 4 \geq \frac{5 \cdot 40b}{2} \log_2 \frac{|\mathcal{L}|}{b} + 120b + 4 \geq \\ &\frac{8 \cdot 5b}{\log_2(4/3)} \log_2 \frac{|\mathcal{L}|}{b} + 24(5b) + 4 \geq 8 \cdot (b + 4) \log_{4/3} \frac{|\mathcal{L}|}{24(b + 4)} + 24(b + 4) + 4 = \\ &8k \log_{4/3} \frac{|\mathcal{L}|}{24k} + 24k + 4, \end{aligned}$$

and thus indeed the preconditions of Lemma 13 are satisfied. Thus, the directed treewidth of $D_i^{(2)}$ is at least $b + 4$. Since $D_i^{(2)}$ is a subgraph of $D_i^{(1)}$ and $\text{dtw}(D_i) \geq \text{dtw}(D_i^{(1)}) - 4$, we have $\text{dtw}(D_i) \geq b$, as required. \blacktriangleleft

5 Wrapping up the Proof of Theorem 5

Proof of Theorem 5. Let G be a directed graph of $\text{dtw}(G) \geq c \cdot a^6 b^8 \log^2(ab)$, where c is a large constant, whose value will follow from the reasoning below. First, we invoke Lemma 7 with $\beta = 2^{37} a^2 b^3 \log(ab)$ and $\alpha = 8ab$ (here we assume that c is sufficiently large so that the assumption is satisfied). We obtain a set of vertex-disjoint paths P_1, \dots, P_{8ab} and sets $A_i, B_i \subseteq V(P_i)$, where A_i appears before B_i on P_i , and $|A_i| = |B_i| = 2^{37} a^2 b^3 \log(ab)$, and the set $\bigcup_{i=1}^{8ab} A_i \cup B_i$ is well-linked. Denote by $\mathcal{L}_{i,j}$ a linkage from B_i to A_j .

72:12 Packing Directed Circuits Quarter-Integrally

We split the $8ab$ paths P_i into a segments, each consisting of $8b$ paths. Formally, for every $\iota \in [a]$ we define $I_\iota = \{j \mid 8(\iota - 1)b < j \leq 8\iota b\}$.

Now we set $r = 64ab^2$ and create an auxiliary r -colored graph H , whose vertices will be paths of appropriately chosen linkages $\mathcal{L}_{i,j}$. More specifically, for every $\iota \in [a]$, and every $i, j \in I_\iota$, we introduce a vertex for every path in $\mathcal{L}_{i,j}$ and color it (i, j) . Two vertices of H are adjacent if and only if their corresponding paths share a vertex in G . Note that for two linkages $\mathcal{L}_{i,j}$ and $\mathcal{L}_{i',j'}$, the graph $H[\mathcal{L}_{i,j} \cup \mathcal{L}_{i',j'}]$ is precisely the intersection graph $I(\mathcal{L}_{i,j}, \mathcal{L}_{i',j'})$.

We set $d := 2^{27}ab \log(ab)$ and consider two cases:

- (i) for all i, j, i', j' the graph $I(\mathcal{L}_{i,j}, \mathcal{L}_{i',j'})$ is d -degenerate.
- (ii) there exist i, j, i', j' , for which the graph $I(\mathcal{L}_{i,j}, \mathcal{L}_{i',j'})$ is not d -degenerate.

An intuition behind case (i) is that for each subgraph of H there is always a path (in G) such that it shares a vertex with at most d paths from all used linkages back.

Case (i) We use Lemma 8 on H . Graph H has $64ab^2$ color classes such that for each $(i, j) \neq (i', j')$ the graph $H[\mathcal{L}_{i,j} \cup \mathcal{L}_{i',j'}]$ is d -degenerate. Note that $|\mathcal{L}_{i,j}| = 2^{37}a^2b^3 \log(ab) \geq 4e(r-1)d$ is sufficiently large to satisfy the last assumption of the lemma. We are given an independent set x_1, \dots, x_{64ab^2} that represents pairwise disjoint paths $L_{i,j}$ from B_i to A_j for all $i, j \in I_\iota$. We also recall that A_i and B_i lie on P_i and all P_i 's are pairwise disjoint.

Let G_ι consist of all paths P_i for $i \in I_\iota$ and $L_{i,j}$ for $i, j \in I_\iota$. By Lemma 9 for $k = 8b$ we obtain $\text{dtw}(G_\iota) \geq b$ while each vertex is in at most 2 such subgraphs. Indeed, each vertex can appear only once on some P_i and once on some $L_{i,j}$.

Case (ii) The claim follows from Lemma 12. Since $|\mathcal{L}| = 2^{37}a^2b^3 \log(ab)$ then $d = 2^{27}ab \log(ab) > 2^{19}ab \log(2^{37}a^2b^2 \log(ab))$. ◀

6 Conclusions

We have shown that if one relaxes the disjointness constraint to quarter-integral packing (i.e., every vertex used at most four times), then the Erdős-Pósa property in directed graphs admits a polynomial bound between the cycle packing number and the feedback vertex set number. A natural question would be to decrease the dependency further, even at the cost of higher congestion (but still a constant). More precisely, we pose the following question: Does there exist a constant c and a polynomial p such that for every integer k if a directed graph G does not contain a family of k cycles such that every vertex of G is in at most c of the cycles, then the directed treewidth of G is at most $kp(\log k)$?

One of the sources of polynomial blow-up in the proof of Theorem 5 is the quadratic blow-up in Lemma 7. Lemma 7 is a direct corollary of another result of [14] that asserts that a directed graph G of directed treewidth $\Omega(k^2)$ contains a path P and a set $A \subseteq V(P)$ that is well-linked and of size k . Is this quadratic blow-up necessary? Can we improve it, even at the cost of some constant congestion in the path P (i.e., allow P to visit every vertex a constant number of times)? We remark that the essence of the improvement from $\mathcal{O}(k^6 \log^2 k)$ (obtained by setting $b = 2$ in Theorem 5) to $\mathcal{O}(k^3)$ asserted by Theorem 4 is to avoid the usage of Lemma 7 and to replace it with a simple well-linkedness trick. However, this trick fails in the general setting of Theorem 5.

References

- 1 Saeed Akhoondian Amiri, Ken-ichi Kawarabayashi, Stephan Kreutzer, and Paul Wollan. The Erdős-Pósa Property for Directed Graphs. *CoRR*, abs/1603.02504, 2016. [arXiv:1603.02504](#).
- 2 Matthew Andrews, Julia Chuzhoy, Venkatesan Guruswami, Sanjeev Khanna, Kunal Talwar, and Lisa Zhang. Inapproximability of edge-disjoint paths and low congestion routing on undirected graphs. *Combinatorica*, 30(5):485–520, 2010.
- 3 Timothy Carpenter, Ario Salmasi, and Anastasios Sidiropoulos. Routing Symmetric Demands in Directed Minor-Free Graphs with Constant Congestion. *CoRR*, abs/1711.01692, 2017. [arXiv:1711.01692](#).
- 4 Chandra Chekuri and Julia Chuzhoy. Large-treewidth graph decompositions and applications. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC 2013)*, pages 291–300. ACM, 2013.
- 5 Chandra Chekuri and Julia Chuzhoy. Polynomial Bounds for the Grid-Minor Theorem. *Journal of the ACM*, 63(5):40:1–40:65, 2016. [doi:10.1145/2820609](#).
- 6 Chandra Chekuri and Alina Ene. The all-or-nothing flow problem in directed graphs with symmetric demand pairs. *Mathematical Programming*, pages 1–24, 2014.
- 7 Chandra Chekuri, Alina Ene, and Marcin Pilipczuk. Constant Congestion Routing of Symmetric Demands in Planar Directed Graphs. *SIAM Journal on Discrete Mathematics*, 32(3):2134–2160, 2018. [doi:10.1137/17M1150694](#).
- 8 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. Multicommodity flow, well-linked terminals, and routing problems. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC 2005)*, pages 183–192. ACM, 2005.
- 9 Julia Chuzhoy and Shi Li. A Polylogarithmic Approximation Algorithm for Edge-Disjoint Paths with Congestion 2. *Journal of the ACM*, 63(5):45:1–45:51, 2016. [doi:10.1145/2893472](#).
- 10 Erik D. Demaine and Mohammadtaghi Hajiaghayi. Linearity of grid minors in treewidth with applications through bidimensionality. *Combinatorica*, 28(1):19–36, 2008.
- 11 Pál Erdős and Lajos Pósa. On independent circuits contained in a graph. *Canadian Journal of Mathematics*, 17:347–352, 1965.
- 12 Meike Hatzel, Ken-ichi Kawarabayashi, and Stephan Kreutzer. Polynomial Planar Directed Grid Theorem. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 1465–1484, 2019.
- 13 Thor Johnson, Neil Robertson, Paul D. Seymour, and Robin Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.
- 14 Ken-ichi Kawarabayashi and Stephan Kreutzer. The Directed Grid Theorem. *CoRR*, abs/1411.5681, 2014. [arXiv:1411.5681](#).
- 15 Ken-ichi Kawarabayashi and Stephan Kreutzer. The Directed Grid Theorem. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC 2015)*, pages 655–664, 2015. [doi:10.1145/2746539.2746586](#).
- 16 Bruce A. Reed. Introducing directed tree width. *Electronic Notes in Discrete Mathematics*, 3:222–229, 1999.
- 17 Bruce A. Reed, Neil Robertson, Paul D. Seymour, and Robin Thomas. Packing directed circuits. *Combinatorica*, 16(4):535–554, 1996.
- 18 Bruce A. Reed and David R. Wood. Polynomial treewidth forces a large grid-like-minor. *European Journal of Combinatorics*, 33(3):374–379, 2012. [doi:10.1016/j.ejc.2011.09.004](#).
- 19 Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984. [doi:10.1016/0095-8956\(84\)90013-3](#).
- 20 Neil Robertson and Paul D. Seymour. Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, 1986. [doi:10.1016/0095-8956\(86\)90030-4](#).
- 21 Neil Robertson, Paul D. Seymour, and Robin Thomas. Quickly Excluding a Planar Graph. *Journal of Combinatorial Theory, Series B*, 62(2):323–348, 1994. [doi:10.1006/jctb.1994.1073](#).
- 22 Paul D. Seymour. Packing Directed Circuits Fractionally. *Combinatorica*, 15(2):281–288, 1995. [doi:10.1007/BF01200760](#).

Equal-Subset-Sum Faster Than the Meet-in-the-Middle

Marcin Mucha 

Institute of Informatics, University of Warsaw, Poland
much@mimuw.edu.pl

Jesper Nederlof 

Eindhoven University of Technology, The Netherlands
j.nederlof@tue.nl

Jakub Pawlewicz 

Institute of Informatics, University of Warsaw, Poland
pan@mimuw.edu.pl

Karol Węgrzycki 

Institute of Informatics, University of Warsaw, Poland
k.wegrzycki@mimuw.edu.pl

Abstract

In the Equal-Subset-Sum problem, we are given a set S of n integers and the problem is to decide if there exist two disjoint nonempty subsets $A, B \subseteq S$, whose elements sum up to the same value. The problem is NP-complete. The state-of-the-art algorithm runs in $\mathcal{O}^*(3^{n/2}) \leq \mathcal{O}^*(1.7321^n)$ time and is based on the *meet-in-the-middle* technique. In this paper, we improve upon this algorithm and give $\mathcal{O}^*(1.7088^n)$ worst case Monte Carlo algorithm. This answers a question suggested by Woeginger in his inspirational survey.

Additionally, we analyse the polynomial space algorithm for Equal-Subset-Sum. A naive polynomial space algorithm for Equal-Subset-Sum runs in $\mathcal{O}^*(3^n)$ time. With read-only access to the exponentially many random bits, we show a randomized algorithm running in $\mathcal{O}^*(2.6817^n)$ time and polynomial space.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms

Keywords and phrases Equal-Subset-Sum, Subset-Sum, meet-in-the-middle, enumeration technique, randomized algorithm

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.73

Related Version A full version of the paper [35] is available at <https://arxiv.org/abs/1905.02424>.

Funding *Marcin Mucha*: Supported by 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).

Jesper Nederlof: Supported by the Netherlands Organization for Scientific Research under project no. 024.002.003 and the European Research Council under project no. 617951.

Karol Węgrzycki: Supported by the grants 2016/21/N/ST6/01468 and 2018/28/T/ST6/00084 of the Polish National Science Center and 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).

Acknowledgements The authors would like to thank anonymous reviewers for their remarks and suggestions. This research has been initiated during Parameterized Algorithms Retreat of University of Warsaw 2019, Karpacz, 25.02-01.03.2019.



© Marcin Mucha, Jesper Nederlof, Jakub Pawlewicz, and Karol Węgrzycki;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 73; pp. 73:1–73:16



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the Subset-Sum problem, we are given as input a set S of n integers a_1, \dots, a_n and a target t . The task is to decide if there exists a subset of S , such that a total sum of the numbers in this subset is equal to t . This can be formulated in the following form:

$$\sum_{i=1}^n x_i a_i = t$$

and the task is to find $x_i \in \{0, 1\}$. Subset-Sum is one of the fundamental NP-complete problems. Study on the exact complexity of Subset-Sum led to the discovery of one of the most fundamental algorithmic tool: *meet-in-the-middle*. [24] used this technique to give a $\mathcal{O}^*(2^{n/2})$ algorithm for Subset-Sum in the following way: First, rewrite the Subset-Sum equation:

$$\sum_{i=1}^{\lfloor n/2 \rfloor} x_i a_i = t - \sum_{i=\lfloor n/2 \rfloor + 1}^n x_i a_i.$$

Then enumerate all $\mathcal{O}(2^{n/2})$ possible values of the left side $L(x_1, \dots, x_{\lfloor n/2 \rfloor})$ and $\mathcal{O}(2^{n/2})$ possible values of the right side $R(x_{\lfloor n/2 \rfloor + 1}, \dots, x_n)$. After that, it remains to look for the value that occurs in both L and R , i.e., *meeting* the tables L and R . One can do that efficiently by sorting (see [24] for details). To summarize, meet-in-the-middle technique is based on rewriting the formula as an equation between two functions and efficiently seeking any value that occurs in both of their images.

Later, [39] observed that space usage of meet-in-the-middle can be improved to $\mathcal{O}^*(2^{n/4})$ by using space-efficient algorithm for 4-SUM. However, the time complexity remains unchallenged and one of the most prominent open problem in the area of exact algorithms is to improve upon *meet-in-the-middle* for Subset-Sum:

► **Open Question 1.** *Can Subset-Sum be solved in $\mathcal{O}^*(2^{(0.5-\delta)n})$ time for some constant $\delta > 0$?*

In this paper, we consider the Equal-Subset-Sum problem. We are given a set S of n integers and the task is to decide if there exist two disjoint nonempty subsets $A, B \subseteq S$, whose elements sum up to the same value. Similarly to Subset-Sum, this problem is NP-complete [45]. In the inspirational survey, [44] noticed Equal-Subset-Sum can be solved by using meet-in-the-middle and asked if it can be improved: ¹

► **Open Question 2** (cf., [43],[44]). *Can we improve upon the meet-in-the-middle algorithm for Equal-Subset-Sum?*

The folklore meet-in-the-middle algorithm for Equal-Subset-Sum (that we will present in the next paragraph) works in $\mathcal{O}^*(3^{n/2})$ time.

Folklore algorithm for Equal-Subset-Sum

First, we arbitrarily partition S into $S_1 = \{a_1, \dots, a_{\lfloor n/2 \rfloor}\}$ and $S_2 = \{a_{\lfloor n/2 \rfloor + 1}, \dots, a_n\}$. Recall that in Equal-Subset-Sum we seek two subsets $A, B \subseteq S$, such that $A \cap B = \emptyset$ and $\Sigma(A) = \Sigma(B)$. We can write the solution as 4 subsets: $A_1 = A \cap S_1$, $A_2 = A \cap S_2$, $B_1 = B \cap S_1$

¹ [43, 44] noticed that 4-SUM gives $\mathcal{O}^*(2^n)$ algorithm, but it actually gives a $\mathcal{O}^*(3^{n/2})$ algorithm, see [35, Appendix C].

and $B_2 = B \cap S_2$, such that: $\Sigma(A_1) + \Sigma(A_2) = \Sigma(B_1) + \Sigma(B_2)$. In particular, it means that: $\Sigma(A_1) - \Sigma(B_1) = \Sigma(B_2) - \Sigma(A_2)$. So, the problem reduces to finding two vectors $x \in \{-1, 0, 1\}^{\lfloor n/2 \rfloor}$ and $y \in \{-1, 0, 1\}^{\lceil n/2 \rceil}$, such that:

$$\sum_{i=1}^{\lfloor n/2 \rfloor} x_i a_i = \sum_{i=1}^{\lceil n/2 \rceil} y_i a_{i+\lfloor n/2 \rfloor}.$$

We can do this in $\mathcal{O}^*(3^{n/2})$ time as follows. First, enumerate and store all $3^{\lfloor n/2 \rfloor}$ possible values of the left side of the equation and all $3^{\lceil n/2 \rceil}$ possible values of the right side of the equation. Then look for a value that occurs in both tables (collision) in time $\mathcal{O}^*(3^{n/2})$ by sorting the values. The total running time is therefore $\mathcal{O}^*(3^{n/2})$. Analogously to Subset-Sum, one can improve the space usage of the above algorithm to $\mathcal{O}^*(3^{n/4})$ (see [35, Appendix C]).

A common pattern seems unavoidable in algorithms for Subset-Sum and Equal-Subset-Sum: we have to go through all possible values of the left and the right side of the equation. This enumeration dominates the time used to solve the problem. So, it was conceivable that perhaps no improvement for Equal-Subset-Sum could be obtained unless we improve an algorithm for Subset-Sum first [43, 44].

1.1 Our Contribution

While the meet-in-the-middle algorithm remains unchallenged for Subset-Sum, we show that, surprisingly, we can improve the algorithm for Equal-Subset-Sum. The main result of this paper is the following theorem.

► **Theorem 1.1.** *Equal-Subset-Sum can be solved in $\mathcal{O}^*(1.7088^n)$ time with high probability.*

This positively answers Open Question 2. To prove this result we observe that the worst case for the meet-in-the-middle algorithm is that of a balanced solution, i.e., when $|A| = |B| = |S \setminus (A \cup B)| \approx n/3$. We propose a substantially different algorithm, that runs in $\mathcal{O}^*(2^{2/3n})$ time for that case. The crucial insight of the new approach is the fact that when $|A| \approx |B| \approx n/3$, then there is an abundance of pairs $X, Y \subseteq S$, $X \neq Y$ with $\Sigma(X) = \Sigma(Y)$. We use the *representation technique* to exploit this. Interestingly, that technique was initially developed to solve the average case Subset-Sum [9, 25].

Our second result is an improved algorithm for Equal-Subset-Sum running in polynomial space. The naive algorithm in polynomial space works in $\mathcal{O}^*(3^n)$ time by enumerating all possible disjoint pairs of subsets of S . This algorithm is analogous to the $\mathcal{O}^*(2^n)$ polynomial space algorithm for Subset-Sum. Recently, [6] proposed a $\mathcal{O}^*(2^{0.86n})$ algorithm for Subset-Sum on the machine that has access to the exponential number of random bits. We show that a similar idea can be used for Equal-Subset-Sum.

► **Theorem 1.2.** *There exists a Monte Carlo algorithm which solves Equal-Subset-Sum in polynomial space and time $\mathcal{O}^*(2.6817^n)$. The algorithm assumes random read-only access to exponentially many random bits.*

This result is interesting for two reasons. First, [6] require nontrivial results in information theory. Our algorithm is relatively simple and does not need such techniques. Second, the approach of [6] developed for Subset-Sum has a barrier, i.e., significantly new ideas must be introduced to get an algorithm running faster than $\mathcal{O}^*(2^{0.75n})$. In our case, this corresponds to the algorithm running in $\mathcal{O}^*(2^{1.5n}) \leq \mathcal{O}^*(2.8285^n)$ time and polynomial space (for elaboration see Section 4). We show that relatively simple observations about Equal-Subset-Sum enable us to give a slightly faster algorithm in polynomial space.

1.2 Related Work

The Equal-Subset-Sum was introduced by [45] who showed that the problem is NP-complete. This reduction automatically excludes $2^{o(n)}$ algorithms for Equal-Subset-Sum assuming ETH (see [35, Appendix B]), hence for this problem we aspire to optimize the constant in the exponent. The best known constant comes from the meet-in-the-middle algorithm. [44] asked if this algorithm for Equal-Subset-Sum can be improved.

Exact algorithms for Subset-Sum

[36] proved that in the exact setting Knapsack and Subset-Sum problems are equivalent.

[39] showed that the meet-in-the-middle algorithm admits a time-space tradeoff, i.e., $\mathcal{T}\mathcal{S}^2 \leq \mathcal{O}^*(2^n)$, where \mathcal{T} is the running time of the algorithm and $\mathcal{S} \leq \mathcal{O}^*(2^{n/2})$ is the space of an algorithm. This tradeoff was improved by [2] for almost all tradeoff parameters.

[3] considered Subset-Sum parametrized by the maximum bin size β and obtained algorithm running in time $\mathcal{O}^*(2^{0.3399n}\beta^4)$. Subsequently, [4] showed that one can get a faster algorithm for Subset-Sum than meet-in-the-middle if $\beta \leq 2^{(0.5-\varepsilon)n}$ or $\beta \geq 2^{0.661n}$. In this paper, we use the hash function that is based on their ideas. Moreover, the ideas in [3, 4] were used in the recent breakthrough polynomial space algorithm [6] running in $\mathcal{O}^*(2^{0.86n})$ time.

From the pseudopolynomial algorithms perspective Knapsack and Subset-Sum admit $\mathcal{O}(nt)$ algorithm, where t is a value of a target. Recently, for Subset-Sum the pseudopolynomial algorithm was improved to run in deterministic $\tilde{\mathcal{O}}(\sqrt{nt})$ time by [29] and randomized $\tilde{\mathcal{O}}(n+t)$ time by [11] (and simplified, see [27, 30]). However, these algorithms have a drawback of running in pseudopolynomial space $\mathcal{O}^*(t)$. Surprisingly, [32] presented an algorithm running in time $\tilde{\mathcal{O}}(n^3t)$ and space $\tilde{\mathcal{O}}(n^2)$ which was later improved to $\tilde{\mathcal{O}}(nt)$ time and $\tilde{\mathcal{O}}(n \log t)$ space assuming the Extended Riemann Hypothesis [11].

From a lower bounds perspective, no algorithm working in $\tilde{\mathcal{O}}(\text{poly}(n)t^{0.99})$ exists for Subset-Sum assuming SETH or SetCover conjecture [18, 1].

Approximation

[45] presented the approximation algorithm for Equal-Subset-Sum with the worst case ratio of 1.324. [7] considered a different formulation of approximation for Equal-Subset-Sum and showed an FPTAS for it.

Cryptography and the average case complexity

In 1978 Knapsack problems were introduced into cryptography by [34]. They introduced a Knapsack based public key cryptosystem. Subsequently, their scheme was broken by using lattice reduction [40]. After that, many knapsack cryptosystems were broken with low-density attacks [31, 17].

More recently, [26] introduced a cryptographic scheme that is provably as secure as Subset-Sum. They proposed a function $f(\vec{a}, S) = \vec{a}, \sum_{i \in S} a_i \pmod{2^{l(n)}}$, i.e., the function which concatenates \vec{a} with the sum of the a_i 's for $i \in S$. Function f is a mapping of an n bit string S to an $l(n)$ bit string and \vec{a} are a fixed parameter. Our algorithms can be thought of as an attempt to find a collision of such a function in the worst case.

However, in the average case more efficient algorithms are known. [42] showed that when solving problems involving sums of elements from lists, one can obtain faster algorithms when there are many possible solutions. In the breakthrough paper, [25] gave $\mathcal{O}^*(2^{0.337n})$

algorithm for an average case Subset-Sum. It was subsequently improved by [9] who gave an algorithm running in $\mathcal{O}^*(2^{0.291n})$. These papers introduced a *representation* technique that is a crucial ingredient in our proofs.

Total search problems

The *Number Balancing* problem is: given n real numbers $a_1, \dots, a_n \in [0, 1]$, find two disjoint subsets $I, J \subseteq [n]$, such that the difference $|\sum_{i \in I} a_i - \sum_{j \in J} a_j|$ is minimized. The pigeonhole principle and the Chebyshev's inequality guarantee that there exists a solution with difference at most $\mathcal{O}(\frac{\sqrt{n}}{2^n})$. [28] showed that in polynomial time one can produce a solution with difference at most $n^{-\Theta(\log n)}$, but since then no further improvement is known.

[37] considered the problem *Equal Sums*: given n positive integers such that their total sum is less than $2^n - 1$, find two subsets with the same sum. By the pigeonhole principle the solution always exists, hence the decision version of this problem is obviously in P. However the hard part is to actually find a solution. Equal Sums is in class PPP but it remains open to show that it is PPP-complete. Recently, this question gained some momentum. [23] showed that Number Balancing is as hard as *Minkowski*. [5] showed the reduction from Equal Sums to Minkowski and conjectured that Minkowski is complete for the class PPP. Very recently, [41] identified the first natural problem complete for PPP.

In [35, Appendix E] we show that our techniques can also be used to solve Number Balancing for integers in $\mathcal{O}^*(1.7088^n)$ time.

Combinatorial Number Theory

If $\Sigma(S) < 2^n - 1$, then by the pigeonhole principle the answer to the decision version of Equal-Subset-Sum on S is always YES. In 1931 Paul Erdős was interested in the smallest maximum value of S , such that the answer to Equal-Subset-Sum on S is NO, i.e., he considered the function:

$$f(n) = \min\{\max\{|S|\} \mid \text{all subsets of } S \text{ are distinct, } |S| = n, S \subseteq \mathbb{N}\}$$

and showed $f(n) > 2^n / (10\sqrt{n})$ [19]. The first nontrivial upper bound on f was $f(n) \leq 2^{n-2}$ (for sufficiently large n) [16]. Subsequently, [33] proved that $f(n) \leq 0.2246 \cdot 2^n$ and [10] showed $f(n) \leq 0.22002 \cdot 2^n$. [20] offered 500 dollars for proof or disproof of conjecture that $f(n) \geq c2^n$ for some constant c .

Other Variants

Equal-Subset-Sum has some connections to the study of the structure of DNA molecules [15, 14, 12]. [13] considered k -Equal-Subset-Sum, in which we need to find k disjoint subsets of a given set with the same sum. They obtained several algorithms that depend on certain restrictions of the sets (e.g., small cardinality of a solution). In the following work, [14] considered other variants of Equal-Subset-Sum and proved their NP-hardness.

2 Preliminaries

Throughout the paper we use the \mathcal{O}^* notation to hide factors polynomial in the input size and the $\tilde{\mathcal{O}}$ notation to hide factors logarithmic in the input size. We also use $[n]$ to denote the set $\{1, \dots, n\}$. If $S = \{a_1, \dots, a_n\}$ is a set of integers and $X \subseteq \{1, \dots, n\}$, then $\Sigma_S(X) := \sum_{i \in X} a_i$. Also, we use $\Sigma(S) = \sum_{s \in S} s$ to denote the sum of the elements of the set. We use the binomial coefficient notation for sets, i.e., for a set S the symbol $\binom{S}{k} = \{X \subseteq S \mid |X| = k\}$ is the set of all subsets of the set S of size exactly k .

We may assume that the input to Equal-Subset-Sum has the following properties:

- the input set $S = \{a_1, \dots, a_n\}$ consists of positive integers,
- $\sum_{i=1}^n a_i < 2^{\tau n}$ for a constant $\tau < 10$,
- integer n is a multiple of 12.

These are standard assumptions for Subset-Sum (e.g., [3, 22]). For completeness, in [35, Appendix A] we prove how to apply reductions to Equal-Subset-Sum to ensure these properties.

We need the following theorem concerning the density of prime numbers [21, p. 371, Eq. (22.19.3)].

► **Lemma 2.1.** *For a large enough integer b , there exist at least $2^b/b$ prime numbers in the interval $[2^b, 2^{b+1}]$.*

The binary entropy function is $h(\alpha) = -\alpha \log_2 \alpha - (1 - \alpha) \log_2 (1 - \alpha)$ for $\alpha \in (0, 1)$ and $h(0) = h(1) = 0$. For all integers $n \geq 1$ and $\alpha \in [0, 1]$ such that σn is an integer, we have the following upper bound on the binomial coefficient [38]: $\binom{n}{\alpha n} \leq 2^{h(\alpha)n}$. We also need a standard bound on binary entropy function $h(x) \leq 2\sqrt{x(1-x)}$.

Throughout this paper all logarithms are base 2.

3 Faster Exponential Space Algorithm

In this section, we improve upon the meet-in-the-middle algorithm for Equal-Subset-Sum.

► **Theorem 3.1.** *Equal-Subset-Sum can be solved in $\mathcal{O}^*(1.7088^n)$ time with high probability.*

Theorem 3.1 is proved by using two different algorithms for Equal-Subset-Sum. To bound the trade-off between these algorithms we introduce the concept of a *minimum solution*.

► **Definition 3.2 (Minimum Solution).** *For a set S of positive integers we say that a solution $A, B \subseteq S$ is a minimum solution if its size $|A| + |B|$ is smallest possible.*

We now assume that the size of the minimum solution has even size for simplicity of presentation. The algorithm and analysis for the case of odd-sized minimum solution is similar, but somewhat more messy due to all the floors and ceilings one needs to take care of.

In Section 3.1 we prove that the meet-in-the-middle approach for Equal-Subset-Sum already gives algorithm running in time $\mathcal{O}^*((3 - \varepsilon)^{n/2})$ if the minimum solution A, B is *unbalanced*, i.e., $||A \cup B| - \frac{2n}{3}| > \varepsilon' n$ for some $\varepsilon' > 0$ depending on ε . Subsequently, in Section 3.2 we propose an algorithm for *balanced* instances, i.e., when the size of a minimum solution is close to $2/3$. In particular, we show how to detect sets A, B with $\Sigma(A) = \Sigma(B)$ and $|A| \approx |B| \approx \frac{n}{3}$, with an $\mathcal{O}^*(2^{\frac{2}{3}n})$ time algorithm. By bounding trade-off between the algorithms from Section 3.1 and Section 3.2 we prove Theorem 3.1 and bound the running time numerically.

3.1 Equal-Subset-Sum for unbalanced solutions via meet-in-the-middle

► **Theorem 3.3.** *If S is a set of n integers with a minimum solution of size ℓ , then Equal-Subset-Sum with input S can be solved in $\mathcal{O}^*\left(\binom{n/2}{\ell/2} 2^{\ell/2}\right)$ time with high probability.*

Algorithm 1 UNBALANCEDEQUALSUBSETSUM(S, ℓ).

- 1: Randomly split S into two disjoint $S_1, S_2 \subseteq S$, such that $|S_1| = |S_2| = n/2$
 - 2: Enumerate $C_1 = \{\Sigma(A_1) - \Sigma(B_1) \mid A_1, B_1 \subseteq S_1, A_1 \cap B_1 = \emptyset, |A_1| + |B_1| = \ell/2\}$
 - 3: Enumerate $C_2 = \{\Sigma(A_2) - \Sigma(B_2) \mid A_2, B_2 \subseteq S_2, A_2 \cap B_2 = \emptyset, |A_2| + |B_2| = \ell/2\}$
 - 4: **if** $\exists x_1 \in C_1, x_2 \in C_2$ such that $x_1 + x_2 = 0$ **then**
 - 5: Let $A_1, B_1 \subseteq S_1$ be such that $x_1 = \Sigma(A_1) - \Sigma(B_1)$
 - 6: Let $A_2, B_2 \subseteq S_2$ be such that $x_2 = \Sigma(A_2) - \Sigma(B_2)$
 - 7: **return** $(A_1 \cup A_2, B_1 \cup B_2)$
 - 8: **end if**
 - 9: **return** NO
-

Proof of Theorem 3.3. Algorithm 1 uses the meet-in-the-middle approach restricted to solutions of size ℓ . We will show that this algorithm solves *Equal – Subset – Sum* in the claimed running time.

The algorithm starts by randomly partitioning the set S into two equally sized sets S_1, S_2 . Let A, B be a fixed minimum solution of size $|A \cup B| = \ell$. We will later show that with $\Omega(1/\text{poly}(n))$ probability $|(A \cup B) \cap S_1| = |(A \cup B) \cap S_2| = \ell/2$. We assume this is indeed the case and proceed with meet-in-the-middle. For S_1 we will list all A_1, B_1 that could possibly be equal to $S_1 \cap A$ and $S_1 \cap B$, i.e. disjoint and with total size $\ell/2$. We compute $x = \Sigma(A_1) - \Sigma(B_1)$ and store all these in C_1 . We proceed analogously for S_2 .

We then look for $x_1 \in C_1$ and $x_2 \in C_2$ such that $x_1 + x_2 = 0$. If we find it then we identify the sets A_1 and B_1 that correspond to x_1 and sets A_2 and B_2 that correspond to x_2 (the easiest way to do that is to store with each element of C_1 and C_2 the corresponding pair of sets when generating them). Finally we return $(A_1 \cup A_2, B_1 \cup B_2)$.

Probability of a good split. We now lower-bound the probability of S_1 and S_2 splitting $A \cup B$ in half. There are $\binom{n}{n/2}$ possible equally sized partitions. Among these there are $\binom{\ell}{\ell/2} \binom{n-\ell}{(n-\ell)/2}$ partitions that split $A \cup B$ in half. The probability that a random partition splits A and B in half is:

$$\frac{\binom{\ell}{\ell/2} \binom{n-\ell}{(n-\ell)/2}}{\binom{n}{n/2}} \geq \frac{2^\ell 2^{n-\ell}}{(n+1)^2 2^n} = \frac{1}{(n+1)^2}$$

because $\frac{2^n}{n+1} \leq \binom{n}{n/2} \leq 2^n$.

Running time. To enumerate C_1 and C_2 we need $\mathcal{O}^*(\binom{n/2}{\ell/2} 2^{\ell/2})$ time, because first we guess set $S_1 \cap (A \cup B)$ of size $\ell/2$ and then split between A and B in at most $2^{\ell/2}$ ways. We then check the existence of $x_1 \in C_1$ and $x_2 \in C_2$ such that $x_1 + x_2 = 0$ in $\mathcal{O}^*((|C_1| + |C_2|) \log(|C_1| + |C_2|))$ time by sorting.

We can amplify the probability of a *good split* to $\mathcal{O}(1)$ by repeating the whole algorithm polynomially many times.

Correctness. With probability $\Omega(1/\text{poly}(n))$ we divide the $A \cup B$ equally between S_1 and S_2 . If that happens the set C_1 contains x_1 such that $x_1 = \Sigma(A \cap S_1) - \Sigma(B \cap S_1)$ and the set C_2 contains x_2 that $x_2 = \Sigma(A \cap S_2) - \Sigma(B \cap S_2)$. Note that $x_1 + x_2 = \Sigma(A \cap S_1) + \Sigma(A \cap S_2) - \Sigma(B \cap S_1) - \Sigma(B \cap S_2) = \Sigma(A) - \Sigma(B)$ which is 0, since A, B is a solution. Therefore Algorithm 1 finds a solution of size ℓ (but of course, it could be different from A, B). ◀

3.2 Equal-Subset-Sum for balanced solutions

► **Theorem 3.4.** *Given a set S of n integers with a minimum solution size $\ell \in (\frac{1}{2}n, (1 - \varepsilon)n]$ for some constant $\varepsilon > 0$, Equal-Subset-Sum can be solved in time $\mathcal{O}^*(2^\ell)$ w.h.p.*

We use Algorithm 2 to prove Theorem 3.4. In this algorithm, we first pick a random prime p in the range $[2^{n-\ell}, 2^{n-\ell+1}]$, as well as an integer t chosen uniformly at random from $[1, 2^{n-\ell}]$. We then compute the set $C = \{X \subseteq S \mid \Sigma(X) \equiv_p t\}$. In the analysis, we argue that with $\Omega(1/\text{poly}(n))$ probability C contains two different subsets X, Y of S with $\Sigma(X) = \Sigma(Y)$. To identify such pair it is enough to sort the set $|C|$ in time $\mathcal{O}(|C| \log |C|)$, and then scan it. We return $X \setminus Y$ and $Y \setminus X$ to guarantee that the returned sets are disjoint.

■ **Algorithm 2** BALANCEDEQUALSUBSETSUM(a_1, \dots, a_n, ℓ).

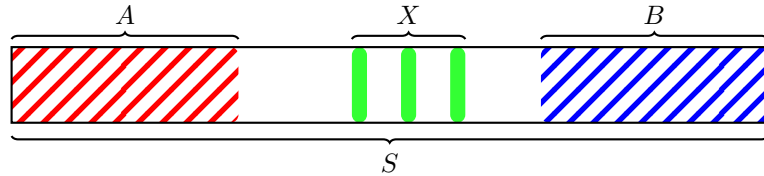
-
- 1: Pick a random prime p in $[2^{n-\ell}, 2^{n-\ell+1}]$
 - 2: Pick a random number t in $[1, 2^{n-\ell}]$
 - 3: Let $C = \{X \subseteq S \mid \Sigma(X) \equiv_p t\}$ be the set of candidates ▷ C contains two sets with equal sum with probability $\Omega(1/\text{poly}(n))$.
 - 4: Enumerate and store all elements of C ▷ In time $\mathcal{O}^*(|C| + 2^{n/2})$
 - 5: Find $X, Y \in C$, such that $\Sigma(X) = \Sigma(Y)$ ▷ In time $\mathcal{O}^*(|C|)$
 - 6: **return** $(X \setminus Y, Y \setminus X)$
-

We now analyse the correctness of Algorithm 2. Later, we will give a bound on the running time and conclude the proof Theorem 3.4. First, observe the following:

► **Lemma 3.5.** *Let S be a set of n positive integers with minimum solution size of ℓ . Let*

$$\Psi = \{\Sigma(X) \mid X \subseteq S \text{ and } \exists Y \subseteq S \text{ such that } X \neq Y \text{ and } \Sigma(X) = \Sigma(Y)\}. \quad (1)$$

If $\ell > \frac{n}{2}$, then $|\Psi| \geq 2^{n-\ell}$ (note that all elements in Ψ are different).



■ **Figure 1** Scheme presents the set S of positive integers and two disjoint subsets $A, B \subseteq S$. The point is that if $\Sigma(A) = \Sigma(B)$ then for any subset $X \subseteq S \setminus (A \cup B)$ we have a guarantee that $\Sigma(A \cup X) = \Sigma(B \cup X)$.

Proof. Let $A, B \subseteq S$ be a fixed minimum solution to S . We know that $\ell = |A \cup B|$, $\Sigma(A) = \Sigma(B)$ and $A \cap B = \emptyset$. With this in hand we construct set Ψ of $2^{n-\ell}$ pairs of different $X, Y \subseteq S$ with $\Sigma(X) = \Sigma(Y)$.

Consider set $Z = S \setminus (A \cup B)$. By the bound on the size of A and B we know that $|Z| = n - \ell$. Now we construct our candidate pairs as follows: take any subset $Z' \subseteq Z$ and note that $X \cup Z'$ and $Y \cup Z'$ satisfy $\Sigma(X \cup Z') = \Sigma(Y \cup Z')$. There are $2^{|Z|}$ possible subsets of set Z and the claim follows.

Now we will prove that if $\ell > \frac{n}{2}$ then all subsets of Z have a different sum. Assume for a contradiction that there exist $Z_1, Z_2 \subseteq Z$, such that $\Sigma(Z_1) = \Sigma(Z_2)$ and $Z_1 \neq Z_2$. Then $Z_1 \setminus Z_2$ and $Z_2 \setminus Z_1$ would give a solution smaller than A, B , because $|Z| < \ell$. This contradicts the assumption about the minimality of A, B . It follows that if $\ell > \frac{1}{2}n$ then all constructed pairs have a different sum. ◀

Now, we consider the hashing function $h_{t,p}(x) = x + t \pmod p$. We prove that if the set Ψ (see Equation 1) is sufficiently large, then for a random choice of t , at least one element of set Ψ is in the congruence class t .

► **Lemma 3.6.** *Let S be the set of n positive integers bounded by $2^{\mathcal{O}(n)}$ with minimum solution of size ℓ and $\ell > \frac{n}{2}$. For a random prime $p \in [2^{n-\ell}, 2^{n-\ell+1}]$ and a random $t \in [1, 2^{n-\ell}]$ let $C_{t,p} = \{X \subseteq S \mid \Sigma(X) \equiv_p t\}$. Then,*

$$\mathbb{P}_{t,p} \left[\exists X, Y \in C_{t,p} \mid \Sigma(X) = \Sigma(Y), X \neq Y \right] \geq \Omega(1/n^2).$$

Proof. Let Ψ be the set defined in (1). So $\Psi \subseteq \{1, \dots, 2^{\mathcal{O}(n)}\}$, and $|\Psi| \geq 2^{n-\ell}$. It is sufficient to bound the probability, that there exists an element $a \in \Psi$ such $a \equiv_p t$. Let $a_1, a_2 \in \Psi$ be two distinct elements.

$$\mathbb{P}_p [a_1 \equiv_p a_2] = \mathbb{P}_p [p \text{ divides } |a_1 - a_2|] \leq \mathcal{O}(n(n - \ell)/2^{n-\ell}).$$

This is because $|a_1 - a_2|$ can only have $\mathcal{O}(n)$ prime divisors, and we are sampling p from the set of at least $2^{n-\ell}/(n - \ell)$ primes by Lemma 2.1. Let k be the number of pairs $a_1, a_2 \in \Psi$ such that $a_1 \equiv_p a_2$. We have $\mathbb{E}[k] \leq \mathcal{O}(|\Psi| + (|\Psi|n)^2/2^{n-\ell})$. We know that $|\Psi| \geq 2^{n-\ell}$, so $\frac{|\Psi|^2}{2^{n-\ell}} \geq |\Psi|$ which means that $\mathbb{E}[k] \leq \mathcal{O}((|\Psi|n)^2/2^{n-\ell})$. Hence, by Markov's inequality k is at most $\mathcal{O}((|\Psi|n)^2/2^{n-\ell})$ with at least constant probability. If this does indeed happen, then

$$|\{a \pmod p \mid a \in \Psi\}| \geq \frac{|\Psi|^2}{k} \geq \Omega\left(\frac{|\Psi|^2}{(|\Psi|n)^2/2^{n-\ell}}\right) \geq \Omega(2^{n-\ell}/n^2),$$

and the probability that t chosen uniformly at random from $[1, 2^{n-\ell}]$ will be among one of the elements of set $\{a \pmod p \mid a \in \Psi\}$ is $|\{a \pmod p \mid a \in \Psi\}|/2^{n-\ell} \geq \Omega(1/n^2)$. ◀

Proof of correctness of Algorithm 2. By Lemma 3.6, after choosing a random prime p and random number $t \in [1, 2^{n-\ell}]$ the set $C = \{X \subseteq S \mid \Sigma(X) \equiv_p t\}$ contains at least two subsets $X, Y \subseteq S$, such that $\Sigma(X) = \Sigma(Y)$ with probability $\Omega(1/\text{poly}(n))$. Algorithm 2 computes the set C and finds $X', Y' \subseteq S$, such that $\Sigma(X') = \Sigma(Y')$. Then it returns the solution $X' \setminus Y', Y' \setminus X'$. ◀

Now we focus on bounding the running time of Algorithm 2. We start by bounding the size of the candidate set C .

▷ **Claim 3.7.** Let S be the set of n non-negative integers bounded by $2^{\mathcal{O}(n)}$ with a minimum solution of size ℓ such that $\ell \leq (1 - \varepsilon)n$ for some constant $\varepsilon > 0$ (think of $\varepsilon = 1/100$). For a random prime $p \in [2^{n-\ell}, 2^{n-\ell+1}]$ and a random number $t \in [1, 2^{n-\ell}]$ let $C_{t,p} = \{X \subseteq S \mid \Sigma(X) \equiv_p t\}$. Then

$$\mathbb{E}[|C_{t,p}|] \leq \mathcal{O}^*(2^\ell)$$

Proof. By the linearity of expectations:

$$\mathbb{E}[|C_{t,p}|] = \sum_{X \subseteq S} \mathbb{P}_{t,p} [p \text{ divides } \Sigma(X) - t]$$

73:10 Equal-Subset-Sum Faster Than the Meet-in-the-Middle

For the remaining part of the proof we focus on showing $\mathbb{P}_{t,p}[p \text{ divides } \Sigma(X) - t] \leq \mathcal{O}^*(2^{\ell-n})$ for a fixed $X \subseteq S$. It automatically finishes the proof, because there are 2^n possible subsets X .

We split the terms into two cases. If $\Sigma(X) = t$, then p divides $\Sigma(X) - t$ with probability 1. However, for a fixed $X \subseteq S$, the probability that $\Sigma(X) = t$ is $\mathcal{O}(\frac{1}{2^{n-\ell}})$ because t is a random number from $[1, 2^{n-\ell}]$ and $p \geq 2^{n-\ell}$.

On the other hand, if $\Sigma(X) \neq t$, then by the assumption, the set S consists of non-negative integers bounded by $2^{\tau n}$ for some constant $\tau > 0$. In particular, $|\Sigma(X) - t| \leq 2^{\tau n}$. This means that $|\Sigma(X) - t|$ has at most $\frac{\tau n}{n-\ell} \leq \frac{\tau}{\varepsilon} = \mathcal{O}(1)$ prime factors of size at least $2^{n-\ell}$. Any prime number p that divides $\Sigma(X) - t$ must therefore be one of these numbers. By Lemma 2.1 there are at least $2^{n-\ell}/(n-\ell)$ prime numbers in range $[2^{n-\ell}, 2^{n-\ell+1}]$. Hence, for a fixed $X \subseteq S$ the probability that p divides $\Sigma(X) - t$ is bounded by $\mathcal{O}(n2^{\ell-n})$. ◀

► **Lemma 3.8.** *The set $C_{t,p}$ can be enumerated in time $\mathcal{O}^*(\max\{|C_{t,p}|, 2^{n/2}\})$.*

The proof of the above lemma is based on [39] algorithm for Subset-Sum. For a full proof of Lemma 3.8 see, e.g., Section 3.2 of [9]. Observe, that for our purposes the running time is dominated by $\mathcal{O}^*(|C_{t,p}|)$.

Proof of the running time of Algorithm 2. To enumerate the set $C_{t,p}$ we need $\mathcal{O}^*(|C| + 2^{n/2})$ time (see Lemma 3.8). To find two subsets $X, Y \in C$, such that $\Sigma(X) = \Sigma(Y)$ we need $\mathcal{O}^*(|C| \log |C|)$ time: we sort C and scan it.

The prime number p is at most $2^{n-\ell+1}$ and the expected size of C is $\mathcal{O}^*(2^\ell)$. Because we assumed that $\ell > \frac{n}{2}$ the expected running time is $\mathcal{O}^*(2^\ell)$ (we can terminate algorithm when it exceeds $\mathcal{O}^*(2^\ell)$ to Monte Carlo guarantees). The probability of success is $\Omega(1/\text{poly}(n))$. We can amplify it with polynomial overhead to any constant by repetition. ◀

This concludes the proof of Theorem 3.4.

3.3 Trade-off for Equal-Subset-Sum

In this section, we will proof the Theorem 3.1 by combining Theorem 3.4 and Theorem 3.3.

Proof of Theorem 3.1. Both Theorem 3.4 and Theorem 3.3 solve Equal-Subset-Sum. Hence, we can focus on bounding the running time. By the trade-off between Theorem 3.4 (which works for $\ell \in (\frac{n}{2}, (1-\varepsilon)n)$) and Theorem 3.3 the running time is:

$$\mathcal{O}^* \left(\max_{\ell \in [1, n/2] \cup [(1-\varepsilon)n, n]} \left\{ \binom{n/2}{\ell/2} 2^{\ell/2} \right\} + \max_{\ell \in (n/2, (1-\varepsilon)n)} \left\{ \min \left\{ \binom{n/2}{\ell/2} 2^{\ell/2}, 2^\ell \right\} \right\} \right)$$

For simplicity of analysis we bounded the sums by the maximum (note that \mathcal{O}^* notation hides polynomial factors). When $\ell \leq n/2$, the running time is maximized for $\ell = n/2$, because (let $\ell = \alpha n$):

$$\mathcal{O}^* \left(\binom{n/2}{\ell/2} 2^{\ell/2} \right) = \mathcal{O}^* \left(2^{\frac{n}{2}(h(\alpha) + \alpha)} \right)$$

and the entropy function $h(x)$ is increasing in range $[0, 0.5)$. For $\ell = \frac{n}{2}$ the running time is $\mathcal{O}^*(2^{0.75n}) \leq \mathcal{O}^*(1.682^n)$. Similarly, we get a running time superior to the claimed one when $\ell \in [(1-\varepsilon)n, n]$. Note that $h(x) \leq 2\sqrt{x(1-x)}$, which means that the running time is bounded by $\mathcal{O}^*(2^{\frac{n}{2}(h(1-\varepsilon) + (1-\varepsilon))}) \leq \mathcal{O}^*(2^{\frac{n}{2}(1+2\sqrt{\varepsilon})})$ which is smaller than our running time for a sufficiently small constant ε .

Finally, when $\ell \in [n/2, (1 - \varepsilon)n]$ we upper bound the running time by the:

$$\mathcal{O}^* \left(\max_{\ell \in [n/2, (1-\varepsilon)n]} \left\{ \min \left\{ 2^{\frac{n}{2}(h(\alpha)+\alpha)}, 2^{\alpha n} \right\} \right\} \right).$$

The above expression is maximized when $h(\alpha) = \alpha$. By numeric calculations $\alpha < 0.77291$, which gives the final running time $\mathcal{O}^*(2^{\alpha n}) \leq \mathcal{O}^*(1.7088^n)$. ◀

4 Polynomial Space Algorithm

The naive algorithm for Equal-Subset-Sum in polynomial space works in $\mathcal{O}^*(3^n)$ time. We are given a set S . We guess a set $A \subseteq S$ and then guess a set $B \subseteq S \setminus A$. Finally, we check if $\Sigma(A) = \Sigma(B)$. The running time is:

$$\mathcal{O}^* \left(\binom{|S|}{|A|} \binom{|S| - |A|}{|B|} \right) \leq \mathcal{O}^*(3^n).$$

Known techniques for Subset-Sum allow us to get an algorithm running in $\mathcal{O}^*(2^{1.5n})$ and polynomial space.

► **Theorem 4.1.** *There exists a Monte Carlo algorithm which solves Equal-Subset-Sum in polynomial space and $\mathcal{O}^*(2^{1.5n}) \leq \mathcal{O}^*(2.8285^n)$ time. The algorithm assumes random read-only access to exponentially many random bits.*

A crucial ingredient of Theorem 4.1 is a nontrivial result for the *Element Distinctness* problem [6, 8]. In this problem, one is given read-only access to the elements of a list $x \in [m]^n$ and the task is to find two different elements of the same value. The problem can be naively solved in $\mathcal{O}(n^2)$ time and $\mathcal{O}(1)$ space by brute force. Also by sorting, we can solve Element Distinctness in $\tilde{\mathcal{O}}(n)$ time and $\tilde{\mathcal{O}}(n)$ space. [8] showed that the problem can be solved in $\tilde{\mathcal{O}}(n^{3/2})$ randomized time and $\tilde{\mathcal{O}}(1)$ space. The algorithm assumes access to a random hash function $f : [m] \rightarrow [n]$.

Proof of Theorem 4.1. We can guarantee random access to the list $L = 2^S$ of all subsets of the set $S = \{a_1, \dots, a_n\}$ on the fly. Namely, for a pointer $x \in \{0, 1\}^n$ we can return an element of the list L that corresponds to x in $\mathcal{O}^*(1)$ time by choosing elements a_i for which $x_i = 1$. More precisely:

$$L(x_1, \dots, x_n) = \{a_i \mid i \in [n], x_i = 1\}.$$

Now to decide Equal-Subset-Sum on set S we execute the Element Distinctness algorithm on the list L of sums of subsets. The list has size 2^n , hence the algorithm runs in $\mathcal{O}^*(2^{1.5n})$ time. Element Distinctness uses only polylogarithmic space in the size of the input, hence our algorithm uses polynomial space. ◀

Quite unexpectedly we can still improve upon this algorithm.

4.1 Improved Polynomial Space Algorithm

In this section, we show an improved algorithm.

► **Theorem 4.2.** *There exists a Monte Carlo algorithm which solves Equal-Subset-Sum in polynomial space and time $\mathcal{O}^*(2.6817^n)$. The algorithm assumes random read-only access to exponentially many random bits.*

73:12 Equal-Subset-Sum Faster Than the Meet-in-the-Middle

Similarly to the exponential space algorithm for Equal-Subset-Sum, we will combine two algorithms. We start with a generalization of Theorem 4.1 parametrized by the size of the solution.

► **Lemma 4.3.** *Let S be a set of n positive integers, $A, B \subseteq S$ be the solution to Equal-Subset-Sum (denote $a = |A|$ and $b = |B|$). There exists a Monte Carlo algorithm which solves Equal-Subset-Sum in polynomial space and time*

$$\mathcal{O}^* \left(\left(\binom{n}{a} + \binom{n}{b} \right)^{1.5} \right).$$

The algorithm assumes random read-only access to exponentially many random bits.

Proof. The proof is just a repetition of the proof of Theorem 4.1 for a fixed sizes of solutions. Our list L will consist of all subsets $\binom{S}{a}$ and $\binom{S}{b}$. Then we run Element Distinctness algorithm, find any sets $A, B \in L$ such that $\Sigma(A) = \Sigma(B)$ and return $A \setminus B, B \setminus A$ to make them disjoint.

The running time follows because Element Distinctness runs in time $\tilde{\mathcal{O}}(n^{1.5})$ and $\text{polylog}(n)$ space. ◀

Note that the runtime of Lemma 4.3 is maximized when $|A| = |B| = n/2$. The next algorithm gives improvement in that case.

► **Lemma 4.4.** *Let S be a set of n positive integers, $A, B \subseteq S$ be the solution to Equal-Subset-Sum (denote $a = |A|$ and $b = |B|$). There exists a Monte Carlo algorithm which solves Equal-Subset-Sum in polynomial space and time*

$$\mathcal{O}^* \left(\min \left\{ \binom{n}{a} 2^{0.75(n-a)}, \binom{n}{b} 2^{0.75(n-b)} \right\} \right).$$

The algorithm assumes random read-only access to exponentially many random bits.

Proof of Lemma 4.4. Without loss of generality, we focus on the case $a \leq b$. First we guess a solution set $A \subseteq S$. We answer YES if we find set $B \subseteq S \setminus A$ such that $\Sigma(A) = \Sigma(B)$ or find two disjoint subsets with equal sum in $S \setminus A$. We show that we can do it in $\mathcal{O}^*(2^{0.75(|S \setminus A|)})$ time and polynomial space which finishes the proof.

First, we arbitrarily partition set $S \setminus A$ into two equally sized sets S_1 and S_2 . Then we create a list $L_1 = [\Sigma(X) \mid X \subseteq S_1]$ and list $L_2 = [\Sigma(A) - \Sigma(X) \mid X \subseteq S_2]$. We do not construct them explicitly because it would take exponential space. Instead we provide a read-only access to them (with the counter technique). We run Element Distinctness on concatenation of L_1 and L_2 . If element distinctness found $x \in L_1$ and $y \in L_2$ such that $x = y$, then we backtrack and look for $X \subseteq S_1$, such that $\Sigma(X) = x$ and $Y \subseteq S_2$, such that $\Sigma(Y) = \Sigma(A) - y$ and return $(A, X \cup Y)$ which is a good solution, because $\Sigma(Y) + \Sigma(X) = \Sigma(A)$.

In the remaining case, i.e. when Element Distinctness finds a duplicate only in one of the lists then, we get a feasible solution as well. Namely, assume that Element Distinctness finds $x, y \in L_1$ such that $x = y$ (the case when $x, y \in L_2$ is analogous). Then we backtrack and look for two corresponding sets $X, Y \subseteq L_1$ such that $X \neq Y$ and $\Sigma(X) = \Sigma(Y) = x$. Finally we return $(X \setminus Y, Y \setminus X)$.

For the running time, note that the size of the list $|L_1| = |L_2| = 2^{0.5|S \setminus A|}$. Hence Element Distinctness runs in time $\mathcal{O}^*((|L_1| + |L_2|)^{1.5}) = \mathcal{O}^*(2^{0.75(n-a)})$. The backtracking takes time $\mathcal{O}^*(|L_1| + |L_2|)$ and polynomial space because we scan through all subsets of S_1 and all subsets of S_2 and look for a set with sum equal to the known value. ◀

Proof of Theorem 4.2. By trade-off between Lemma 4.4 and Lemma 4.3 we get the following running time:

$$\mathcal{O}^* \left(\max_{1 \leq a, b \leq n} \left\{ \min \left\{ \left(\binom{n}{a} + \binom{n}{b} \right)^{1.5}, \binom{n}{a} 2^{0.75(n-a)}, \binom{n}{b} 2^{0.75(n-b)} \right\} \right\} \right)$$

By symmetry this expression is maximized when $a = b$. Now we will write the exponents by using entropy function (let $a = \alpha n$):

$$\mathcal{O}^* \left(\max_{\alpha \in [0,1]} \left\{ \min \left\{ 2^{1.5h(\alpha)n}, 2^{(h(\alpha)+0.75(1-\alpha))n} \right\} \right\} \right)$$

The expression is maximized when $1.5h(\alpha) = h(\alpha) + 0.75(1 - \alpha)$, By numerical computations $\alpha < 0.36751$, which means that the running time is $\mathcal{O}^*(2^{1.42312n}) \leq \mathcal{O}^*(2.6817^n)$. ◀

5 Conclusion and Open Problems

In this paper, we break two natural barriers for Equal-Subset-Sum: we propose an improvement upon the meet-in-the-middle algorithm and upon the polynomial space algorithm. Our techniques have additional applications in the problem of finding collision of hash function in cryptography and the number balancing problem (see [35, Appendix E]).

We believe that our algorithms can potentially be improved with more involved techniques. However, getting close to the running time of Subset-Sum seems ambitious. In [35, Appendix B] we show that a faster algorithm than $\mathcal{O}^*(1.1893^n)$ for Equal-Subset-Sum would yield a faster than $\mathcal{O}^*(2^{n/2})$ algorithm for Subset-Sum. It is quite far from our bound $\mathcal{O}^*(1.7088^n)$. The main open problem is therefore to close the gap between upper and lower bounds for Equal-Subset-Sum.

References

- 1 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-Based Lower Bounds for Subset Sum and Bicriteria Path. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 41–57. SIAM, 2019.
- 2 Per Austrin, Petteri Kaski, Mikko Koivisto, and Jussi Määtä. Space-Time Tradeoffs for Subset Sum: An Improved Worst Case Algorithm. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 45–56. Springer, 2013.
- 3 Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Subset Sum in the Absence of Concentration. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPICs*, pages 48–61. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.
- 4 Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Dense Subset Sum May Be the Hardest. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPICs*, pages 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
- 5 Frank Ban, Kamal Jain, Christos H. Papadimitriou, Christos-Alexandros Psomas, and Aviad Rubinfeld. Reductions in PPP. *Inf. Process. Lett.*, 145:48–52, 2019.

- 6 Nikhil Bansal, Shashwat Garg, Jesper Nederlof, and Nikhil Vyas. Faster space-efficient algorithms for subset sum and k-sum. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 198–209. ACM, 2017.
- 7 Cristina Bazgan, Miklos Santha, and Zs. Tuza. Efficient approximation algorithms for the Subset-Sums Equality problem. In *International Colloquium on Automata, Languages, and Programming*, pages 387–396. Springer, 1998.
- 8 Paul Beame, Raphaël Clifford, and Widad Machmouchi. Element Distinctness, Frequency Moments, and Sliding Windows. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 290–299. IEEE Computer Society, 2013.
- 9 Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved Generic Algorithms for Hard Knapsacks. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 364–385. Springer, 2011.
- 10 Tom Bohman. A sum packing problem of Erdős and the Conway-Guy sequence. *Proceedings of the American Mathematical Society*, 124(12):3627–3636, 1996.
- 11 Karl Bringmann. A Near-linear Pseudopolynomial Time Algorithm for Subset Sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '17*, pages 1073–1084, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics.
- 12 Mark Cieliebak. *Algorithms and hardness results for DNA physical mapping, protein identification, and related combinatorial problems*. PhD thesis, ETH Zürich, 2003.
- 13 Mark Cieliebak, Stephan Eidenbenz, and Aris Pagourtzis. Composing equipotent teams. In *International Symposium on Fundamentals of Computation Theory*, pages 98–108. Springer, 2003.
- 14 Mark Cieliebak, Stephan Eidenbenz, Aris Pagourtzis, and Konrad Schlude. On the Complexity of Variations of Equal Sum Subsets. *Nord. J. Comput.*, 14(3):151–172, 2008.
- 15 Mark Cieliebak, Stephan Eidenbenz, and Paolo Penna. Noisy data make the partial digest problem NP-hard. In *International Workshop on Algorithms in Bioinformatics*, pages 111–123. Springer, 2003.
- 16 John H Conway and Richard K Guy. Sets of natural numbers with distinct subset sums. *Notices Amer. Math. Soc.*, 15:345, 1968.
- 17 Matthijs J. Coster, Antoine Joux, Brian A. LaMacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. Improved Low-Density Subset Sum Algorithms. *Computational Complexity*, 2:111–128, 1992.
- 18 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 Conference on Computational Complexity, CCC 2012, Porto, Portugal, June 26-29, 2012*, pages 74–84. IEEE Computer Society, 2012.
- 19 Paul Erdős. Problems and results in additive number theory. *Journal London Wash. Soc.*, 16:212–215, 1941.
- 20 Paul Erdős. A survey of problems in combinatorial number theory. *Annals of Discrete Mathematics*, 6:89–115, 1980.
- 21 Godfrey Harold Hardy, Edward Maitland Wright, et al. *An introduction to the theory of numbers*. Oxford university press, 1979.
- 22 Danny Harnik and Moni Naor. On the Compressibility of NP Instances and Cryptographic Applications. *SIAM J. Comput.*, 39(5):1667–1713, 2010.
- 23 Rebecca Hoberg, Harishchandra Ramadas, Thomas Rothvoss, and Xin Yang. Number Balancing is as Hard as Minkowski’s Theorem and Shortest Vector. In Friedrich Eisenbrand and Jochen Köneemann, editors, *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, volume 10328 of *Lecture Notes in Computer Science*, pages 254–266. Springer, 2017.

- 24 Ellis Horowitz and Sartaj Sahni. Computing Partitions with Applications to the Knapsack Problem. *J. ACM*, 21(2):277–292, 1974.
- 25 Nick Howgrave-Graham and Antoine Joux. New Generic Algorithms for Hard Knapsacks. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 235–256. Springer, 2010.
- 26 Russell Impagliazzo and Moni Naor. Efficient Cryptographic Schemes Provably as Secure as Subset Sum. *J. Cryptology*, 9(4):199–216, 1996.
- 27 Ce Jin and Hongxun Wu. A Simple Near-Linear Pseudopolynomial Time Randomized Algorithm for Subset Sum. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, volume 69 of *OASICS*, pages 17:1–17:6. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019.
- 28 Narendra Karmarkar and Richard M. Karp. An Efficient Approximation Scheme for the One-Dimensional Bin-Packing Problem. In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 312–320. IEEE Computer Society, 1982.
- 29 Konstantinos Koiliaris and Chao Xu. A Faster Pseudopolynomial Time Algorithm for Subset Sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 1062–1072, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics.
- 30 Konstantinos Koiliaris and Chao Xu. Subset Sum Made Simple. *CoRR*, abs/1807.08248, 2018. [arXiv:1807.08248](https://arxiv.org/abs/1807.08248).
- 31 J. C. Lagarias and Andrew M. Odlyzko. Solving Low-Density Subset Sum Problems. *J. ACM*, 32(1):229–246, 1985.
- 32 Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 321–330. ACM, 2010.
- 33 W Fred Lunnon. Integer sets with distinct subset-sums. *Mathematics of Computation*, 50(181):297–320, 1988.
- 34 Ralph C. Merkle and Martin E. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Trans. Information Theory*, 24(5):525–530, 1978.
- 35 Marcin Mucha, Jesper Nederlof, Jakub Pawlewicz, and Karol Węgrzycki. Equal-Subset-Sum Faster Than the Meet-in-the-Middle. *CoRR*, abs/1905.02424, 2019. [arXiv:1905.02424](https://arxiv.org/abs/1905.02424).
- 36 Jesper Nederlof, Erik Jan van Leeuwen, and Ruben van der Zwaan. Reducing a Target Interval to a Few Exact Queries. In Branislav Rován, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 of *Lecture Notes in Computer Science*, pages 718–727. Springer, 2012.
- 37 Christos H. Papadimitriou. On the Complexity of the Parity Argument and Other Inefficient Proofs of Existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.
- 38 Herbert Robbins. A remark on Stirling’s formula. *The American mathematical monthly*, 62(1):26–29, 1955.
- 39 Richard Schroepel and Adi Shamir. $A \cdot T = O(2^{n/2})$. *SIAM J. Comput.*, 10(3):456–464, 1981.
- 40 Adi Shamir. A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem. *IEEE Trans. Information Theory*, 30(5):699–704, 1984.
- 41 Katerina Sotiraki, Manolis Zampetakis, and Giorgos Zirdelis. PPP-Completeness with Connections to Cryptography. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 148–158. IEEE Computer Society, 2018.

73:16 Equal-Subset-Sum Faster Than the Meet-in-the-Middle

- 42 David A. Wagner. A Generalized Birthday Problem. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002.
- 43 Gerhard J. Woeginger. Space and Time Complexity of Exact Algorithms: Some Open Problems (Invited Talk). In Rodney G. Downey, Michael R. Fellows, and Frank K. H. A. Dehne, editors, *Parameterized and Exact Computation, First International Workshop, IWPEC 2004, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3162 of *Lecture Notes in Computer Science*, pages 281–290. Springer, 2004.
- 44 Gerhard J. Woeginger. Open problems around exact algorithms. *Discrete Applied Mathematics*, 156(3):397–405, 2008.
- 45 Gerhard J. Woeginger and Zhongliang Yu. On the Equal-Subset-Sum Problem. *Inf. Process. Lett.*, 42(6):299–302, 1992.


Hardness of Bichromatic Closest Pair with Jaccard Similarity

Rasmus Pagh 

BARC, Copenhagen, Denmark
IT University of Copenhagen, Denmark
pagh@itu.dk

Nina Mesing Stausholm 

BARC, Copenhagen, Denmark
IT University of Copenhagen, Denmark
nimn@itu.dk

Mikkel Thorup 

BARC, Copenhagen, Denmark
University of Copenhagen, Denmark
mikkel2thorup@gmail.com

Abstract

Consider collections \mathcal{A} and \mathcal{B} of red and blue sets, respectively. Bichromatic Closest Pair is the problem of finding a pair from $\mathcal{A} \times \mathcal{B}$ that has similarity higher than a given threshold according to some similarity measure. Our focus here is the classic Jaccard similarity $|\mathbf{a} \cap \mathbf{b}|/|\mathbf{a} \cup \mathbf{b}|$ for $(\mathbf{a}, \mathbf{b}) \in \mathcal{A} \times \mathcal{B}$.

We consider the approximate version of the problem where we are given thresholds $j_1 > j_2$ and wish to return a pair from $\mathcal{A} \times \mathcal{B}$ that has Jaccard similarity higher than j_2 if there exists a pair in $\mathcal{A} \times \mathcal{B}$ with Jaccard similarity at least j_1 . The classic locality sensitive hashing (LSH) algorithm of Indyk and Motwani (STOC '98), instantiated with the MinHash LSH function of Broder et al., solves this problem in $\tilde{O}(n^{2-\delta})$ time if $j_1 \geq j_2^{1-\delta}$. In particular, for $\delta = \Omega(1)$, the approximation ratio $j_1/j_2 = 1/j_2^\delta$ increases polynomially in $1/j_2$.

In this paper we give a corresponding hardness result. Assuming the Orthogonal Vectors Conjecture (OVC), we show that there cannot be a general solution that solves the Bichromatic Closest Pair problem in $O(n^{2-\Omega(1)})$ time for $j_1/j_2 = 1/j_2^{\epsilon(1)}$. Specifically, assuming OVC, we prove that for any $\delta > 0$ there exists an $\epsilon > 0$ such that Bichromatic Closest Pair with Jaccard similarity requires time $\Omega(n^{2-\delta})$ for any choice of thresholds $j_2 < j_1 < 1 - \delta$, that satisfy $j_1 \leq j_2^{1-\epsilon}$.

2012 ACM Subject Classification Theory of computation \rightarrow Problems, reductions and completeness

Keywords and phrases fine-grained complexity, set similarity search, bichromatic closest pair, jaccard similarity

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.74

Related Version A full version of the paper is available at <https://arxiv.org/abs/1907.02251>.

Funding This work was supported by Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.

Rasmus Pagh: This research has received funding from the European Research Council under the European Union's 7th Framework Programme (FP7/2007-2013) / ERC grant agreement no. 614331.

Acknowledgements We want to thank A. Rubinfeld for helping us understand the background of his results in [9].

1 Introduction

Twitter is a well-known social network, in which a user can connect to other users by *following* them [5]. Users can read and write messages called *tweets* of up to 280 characters. An important service that Twitter provides is helping users discover other users that they might



© Rasmus Pagh, Nina M. Stausholm, and Mikkel Thorup;
licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 74; pp. 74:1–74:13



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

like to follow, by making suggestions. This service is called the *You might also want to follow*-service and is better known as the WTF (Who To Follow) recommender system [6]. In order to suggest connections that the user might like, they should be similar to the user's existing connections. As an example, if a user is already connected to Cristiano Ronaldo, Twitter might suggest Lionel Messi as a new connection, since the connection to Ronaldo hints that the user likes famous soccer players. Hence, we need a way to decide if a connection is similar to an existing connection. We might for instance suggest a new connection if the tweets are similar to the tweets of an existing connection or if the connection has a lot of the same followers as an existing connection.

The main challenge is to find similar connections when the number of user accounts increases drastically and the task is particularly difficult when the similarity does not need to be significant, i.e., when we look for connections that have only little in common with existing ones, while they may still be of interest to the particular user [5]. This leads us to the notion of *similarity search*, which concerns the general problem of searching for similar objects in a collections of objects. Often we consider these objects as sets representing some concept or entity. An object could for example be a document that is represented by a set of words. Hence, we talk about *set similarity search*.

There are several versions of the problem addressing different situations. In this paper we consider a batched version of set similarity search, namely the Bichromatic Closest Pair which can be informally described as follows:

Suppose we are given collections \mathcal{A} and \mathcal{B} , each of n sets from a universe of size $O(\log n)$. We refer to the sets in \mathcal{A} as *red* and the sets in \mathcal{B} as *blue*. Bichromatic Closest Pair is the problem of finding the pair consisting of a red and a blue set that is closest with respect to some distance or similarity measure. We will concern ourselves with Jaccard similarity, which is defined for a pair of sets $(\mathbf{a}, \mathbf{b}) \in \mathcal{A} \times \mathcal{B}$ as

$$J(\mathbf{a}, \mathbf{b}) = \frac{|\mathbf{a} \cap \mathbf{b}|}{|\mathbf{a} \cup \mathbf{b}|} = \frac{|\mathbf{a} \cap \mathbf{b}|}{|\mathbf{a}| + |\mathbf{b}| - |\mathbf{a} \cap \mathbf{b}|}. \quad (1)$$

In particular, we consider the following *decision version* of Bichromatic Closest Pair with Jaccard similarity: decide whether there exists a pair $(\mathbf{a}, \mathbf{b}) \in \mathcal{A} \times \mathcal{B}$ such that $J(\mathbf{a}, \mathbf{b}) \geq j_1$ or if all pairs $(\mathbf{a}, \mathbf{b}) \in \mathcal{A} \times \mathcal{B}$, has $J(\mathbf{a}, \mathbf{b}) < j_2$ for given thresholds j_1 and j_2 .

It is well-known that we can solve Bichromatic Closest Pair with Jaccard similarity for thresholds satisfying $j_1 \geq j_2^{1-\delta}$ in time $O(n^{2-\delta})$ (see Section 1.1). In particular, for $\delta = \Omega(1)$, the approximation ratio $j_1/j_2 = 1/j_2^\delta$ increases polynomially in $1/j_2$. In this paper, we will present a corresponding hardness result. The hardness is conditioned on one of the most well-known and widely believed hypotheses, namely the Orthogonal Vectors Conjecture [11].

► **Conjecture 1** (Orthogonal Vectors Conjecture (OVC)). *For every $\delta > 0$ there exists $c = c(\delta)$ such that given two collections $\mathcal{A}, \mathcal{B} \subset \{0, 1\}^m$ of cardinality n , where $m = c \log n$, deciding if there is a pair $(\mathbf{a}, \mathbf{b}) \in \mathcal{A} \times \mathcal{B}$ such that $\mathbf{a} \cdot \mathbf{b} = 0$ requires time $\Omega(n^{2-\delta})$.*

Assuming OVC, we show that there cannot be a general solution that solves the Bichromatic Closest Pair problem with Jaccard similarity in $O(n^{2-\Omega(1)})$ time for $j_1/j_2 = 1/j_2^{o(1)}$. More specifically, we show

► **Theorem 2.** *Assuming the Orthogonal Vectors Conjecture (OVC), the following holds: for any $\delta > 0$, there exists an $\varepsilon > 0$ such that for any given $j_2 < j_1 < 1 - \delta$ satisfying $j_1 \leq j_2^{1-\varepsilon}$, solving Bichromatic Closest Pair with Jaccard similarity for n red and n blue sets for sets from a universe of size $\ln(n)/j_2^{O(\log 1/j_1)}$ for thresholds j_1 and j_2 requires time $\Omega(n^{2-\delta})$.*

The dependence of ε on δ is unspecified because the function $c(\delta)$ in OVC is not specified, see discussion in Appendix B in the full version on ArXiv [8, App. B].

1.1 Techniques and Related Work

Similarity search can be performed in several ways – a popular technique is Locality Sensitive Hashing (LSH) [7] which attempts to collect similar items in buckets in order to reduce the number of sets needed to check similarity against. We can for example use Broder’s MinHash [1] with locality sensitive hashing to solve Bichromatic Closest Pair with Jaccard similarity in time $\tilde{O}(n^{2-\varepsilon})$ when $j_1 \geq j_2^{1-\varepsilon}$ for any ε . This is done by ensuring that the collision probability for pairs with similarity j_2 is $1/n$ and the collision probability for pairs with similarity j_1 is $1/n^{1-\varepsilon}$. Hashing $n^{1-\varepsilon}$ times means that we find a pair with similarity j_1 if one exists. The ChosenPath method presented in [4] also uses the LSH framework to solve Bichromatic Closest Pair with Braun-Blanquet similarity in time $\tilde{O}(n^{2-\varepsilon})$ for thresholds $j_1 \geq j_2^{1-\varepsilon}$.

The proof of Theorem 2 will be based on a result by Rubinfeld [9]: Assuming the Orthogonal Vectors Conjecture, a $(1 + \varepsilon)$ -approximation to Bichromatic Closest Pair with Hamming, Edit or Euclidean distance requires time $\Omega(n^{2-\delta})$. The required approximation factor $1 + \varepsilon$ depends on δ , and tends to 1 as δ tends to zero. We translate this into an equivalent conditional lower bound for Jaccard similarity for certain constants j_1 and j_2 .

In order to handle smaller subconstant values of j_1 and j_2 we use a technique that we call squaring, which allows us to increase the gap in similarities between pairs with high Jaccard similarity and pairs with low Jaccard similarity by computing the cartesian product of a binary vector with itself. A similar technique is used in [10] by Valiant. His technique is called *tensoring* and is used to amplify the gap between small and large inner products of vectors. We also see a similar technique in the LSH framework with MinHash, where we use concatenation of hash values (which are sampled set elements) to amplify the difference in collision probability, and hence in the Jaccard similarity.

Combining two simple reductions with the above squaring we show that for any δ , we can always find ε such that Bichromatic Closest Pair with Jaccard similarity cannot be solved in time $O(n^{2-\delta})$ for any pair $j_1, j_2 < 1 - \delta$ when $j_1 \leq j_2^{1-\varepsilon}$. Contrast this with the above LSH upper bound of $\tilde{O}(n^{2-\delta})$ for $j_1 \geq j_2^{1-\delta}$. We also know that there are parts of the parameter space where $j_1 = j_2^{1-\delta}$ that can be solved in $\tilde{O}(n^{2-\delta-\Omega(1)})$ time, see the discussion in [4]. While LSH with MinHash is not the fastest possible algorithm in terms of the exponent achieved, it has been unclear how far from optimal it might be.

Other related work

Very recently, Chen and Williams [3] showed that assuming the OVC we cannot additively approximate our Bichromatic Closest Pair problem with Jaccard similarity. It might be possible to use Chen and Williams as a base for showing our main theorem, but this would require reductions quite different from the ones presented in this paper.

An earlier result of Chen [2] shows that it is not possible (under OVC) to compute a $(d/\log n)^{o(1)}$ -approximation to Maximum Inner Product (Max-IP) with two sets of n vectors from $\{0, 1\}^d$ in time $O(n^{2-\Omega(1)})$.

2 Preliminaries

2.1 Notation

We will occasionally consider a set, \mathbf{x} , from a finite universe $U = \{u_1, \dots, u_{|U|}\}$ as a vector \mathbf{v} of dimension $|U|$ such that $v_i = [u_i \in \mathbf{x}]$, in Iverson notation. We call this vector the *characteristic vector for \mathbf{x}* . Hence, we refer to the set of indexes and the universe interchangeably. We denote the Hamming weight of a binary vector \mathbf{v} by $|\mathbf{v}|$. In the following, we will not only index vectors with integers, but also with vectors of integers. Hence, we will consider vectors of dimension d^2 with entries v_{ij} , for $i = (i_1, \dots, i_d)$ and $j = (j_1, \dots, j_d)$.

2.2 Bichromatic Closest Pair

Recall Jaccard similarity as is defined in (1). We define Bichromatic Closest Pair with Jaccard similarity for thresholds t_1 and t_2 as follows: Let U be a universe of size $O(\log n)$. Given collections \mathcal{A} and \mathcal{B} , each of n sets from U , and thresholds $t_2 < t_1 < 1$, we will consider the problem of finding a pair of sets $(\mathbf{a}, \mathbf{b}) \in \mathcal{A} \times \mathcal{B}$ with $J(\mathbf{a}, \mathbf{b}) \geq t_2$ if there exists a pair $(\mathbf{a}^*, \mathbf{b}^*) \in \mathcal{A} \times \mathcal{B}$ with $J(\mathbf{a}^*, \mathbf{b}^*) \geq t_1$. If all pairs have $J(\mathbf{a}, \mathbf{b}) < t_2$, we must not return any pair of sets.

2.3 Useful instances of Bichromatic Closest Pair

The following lemma corresponds to Theorem 4.1 in [9] and will form the basis of our results. It includes the important properties of the instances constructed in the proof the theorem, which we will use actively to prove our own Theorem 2.

► **Lemma 3.** *Assume OVC. Given $\delta > 0$, there exist $\varepsilon > 0$ and values h_1, h_2 where $h_2 = (1 + \varepsilon)h_1$ such that Bichromatic Closest Pair with Hamming distance for thresholds h_1 and h_2 requires time $\Omega(n^{2-\delta})$ for instances with n red and n blue sets from a universe of size $O(\log n)$. There are instances that require this time with the following properties, where we let $T = O(\frac{1}{\varepsilon})$ and $m = O(\log n)$:*

- All red sets have size Tm and all blue sets have size m .
- The thresholds h_1 and h_2 are $m(T - 1)$ and mT , respectively.
- All sets in the instance come from a universe of size $2Tm$.

In particular, the lemma states that we cannot compute a $(1 + \varepsilon)$ -approximation to Bichromatic Closest Pair with Hamming distance in truly subquadratic time. We will extend this result in a few steps, using the properties of the hard instances, to achieve Theorem 2.

2.4 Hardness of Bichromatic Closest Pair with Jaccard similarity

In order to prove Theorem 2, we need the following lemma, which extends Lemma 3 in the natural way to Jaccard similarity.

► **Lemma 4.** *Assuming OVC, we have the following: For any $\delta > 0$ there exist j_1, j_2 with $j_1 = 2 \cdot j_2$ such that Bichromatic Closest Pair with Jaccard similarity with thresholds j_1 and j_2 requires time $\Omega(n^{2-\delta})$.*

Proof. We use instances as described in Lemma 3. First, note that

$$J(\mathbf{a}, \mathbf{b}) = \frac{|\mathbf{a} \cap \mathbf{b}|}{|\mathbf{a} \cup \mathbf{b}|} = \frac{\frac{|\mathbf{a}| + |\mathbf{b}| - d_H(\mathbf{a}, \mathbf{b})}{2}}{|\mathbf{a}| + |\mathbf{b}| - \frac{|\mathbf{a}| + |\mathbf{b}| - d_H(\mathbf{a}, \mathbf{b})}{2}} = \frac{|\mathbf{a}| + |\mathbf{b}| - d_H(\mathbf{a}, \mathbf{b})}{|\mathbf{a}| + |\mathbf{b}| + d_H(\mathbf{a}, \mathbf{b})}$$

which implies that letting

$$j_1 = \frac{Tm + m - m(T - 1)}{Tm + m + m(T - 1)} = \frac{1}{T} \quad \text{and} \quad j_2 = \frac{Tm + m - Tm}{Tm + m + Tm} = \frac{1}{2T + 1},$$

we cannot solve Bichromatic Closest Pair with Jaccard similarity in time $O(n^{2-\delta})$. Since $T = O(\frac{1}{\varepsilon})$, as mentioned in Lemma 3, we get a lower bound for the approximation factor:

$$\frac{\frac{1}{T}}{\frac{1}{2T+1}} = \frac{2T+1}{T} = 2 + \frac{1}{T} = 2 + \Omega(\varepsilon).$$

In particular, we achieve hardness of a 2-approximation. ◀

3 Overview of reductions used

We prove Theorem 2 by combining several reductions into one. So let $(\mathcal{A}, \mathcal{B})$ be any instance of Bichromatic Closest Pair with Jaccard similarity as described in Lemma 3. We give a brief introduction to each of these reductions – note that all reductions are self-reductions. We give the details of the proof and the use of each reduction in Section 5. Further details can be found in Appendix B in the full version on ArXiv [8, App. B].

- **Adding common elements to sets:** Adding common elements to all sets in collections \mathcal{A} and \mathcal{B} increases the Jaccard similarity between any pair of red and blue sets.
- **Adding different elements to sets:** Adding elements to all sets in \mathcal{A} decreases the Jaccard similarity between any pair of red and blue sets.
- **Squaring:** Consider all sets by their characteristic vector. We define squaring as follows: given vector $\mathbf{a} = (a_1, \dots, a_d)$ the squared vector has entries

$$a'_{ij} = a_i \cdot a_j \quad \text{for } i, j \in \{1, \dots, d\}.$$

The resulting vector \mathbf{a}' , which is the characteristic vector for $\mathbf{a} \times \mathbf{a}$, has dimension d^2 as described in Section 2.1. Vector \mathbf{a}' can equivalently be considered as a set from a universe of size d^2 . We will use this reduction iteratively to reduce the Jaccard similarity between any pair of vectors in the instance of Bichromatic Closest Pair.

- **Sampling:** We will use sampling to reduce the size of the universe after each step of squaring. Hence, we consider squaring and sampling as a single reduction which first squares the vectors and then samples from the resulting vectors. We will use the squaring-and-sampling reduction iteratively.

4 The squaring-and-sampling reduction – details

In the proof of Theorem 2 we will take any instance of Bichromatic Closest Pair with Jaccard similarity with the properties described in Lemma 3 and use the squaring reduction described in Section 3 to decrease the Jaccard similarity of every pair of sets in the instance. We will argue that a solution for the new instance also provides a solution for the original instance. When squaring all sets, the Jaccard similarity between any pair of sets will decrease, so we need to capture this change in the thresholds, such that a solution for the new instance implies a solution for the initial instance. When squaring the sets in \mathcal{A} and \mathcal{B} , the size of the sets will be squared and it is easy to see that so will the size of the intersection. Hence, the Jaccard similarity of a pair (\mathbf{a}, \mathbf{b}) after squaring i times, $(\mathbf{a}_i, \mathbf{b}_i)$ is

$$J(\mathbf{a}_i, \mathbf{b}_i) = \frac{|\mathbf{a} \cap \mathbf{b}|^{2^i}}{|\mathbf{a}|^{2^i} + |\mathbf{b}|^{2^i} - |\mathbf{a} \cap \mathbf{b}|^{2^i}}. \quad (2)$$

In order to keep down the size of the universe, we need to sample after each step of squaring. This might incur a small error in the Jaccard similarity. The next few sections will bound this error. From this point, we will denote the squaring-and-sampling reduction by f . Hence, applying the reduction f to a set, \mathbf{v} , i times will yield a set $\mathbf{f}(\mathbf{v}, \mathbf{i})$.

4.1 Subsampling

We bound the error incurred in each of $|\mathbf{a} \cap \mathbf{b}|$, $|\mathbf{a}|$ and $|\mathbf{b}|$ and combine these with a union bound to get a bound on the error in the Jaccard similarity. We shall see that when sampling sufficiently many elements from the universe the sets are taken from, we get that with high probability a solution for the constructed instance will provide a valid solution for the original instance.

74:6 Hardness of Bichromatic Closest Pair with Jaccard Similarity

The following lemmas will help us show that sampling after squaring will not distort the similarity of the resulting vectors too much.

► **Lemma 5.** *Let $0 < m' < m < 1$ and let \mathbf{p} be a set from a universe of size s^2 for an integer s . Assume that $(m' \cdot s)^2 \leq |\mathbf{p}| \leq (m \cdot s)^2$. Sample s' elements from the universe uniformly at random, \mathbf{z} , thus generating sample set $\mathbf{p} \cap \mathbf{z}$. We have*

$$(1 - \gamma) \cdot m'^2 \cdot s' \leq |\mathbf{p} \cap \mathbf{z}| \leq (1 + \gamma) \cdot m^2 \cdot s'$$

with probability at least $1 - 2n^{-10}$ when sampling $s' \geq \frac{30 \ln(n)}{\gamma^2 m'^2}$ elements.

Proof. The result is an immediate consequence of the Chernoff bound: when we sample $s' \geq \frac{30 \ln(n)}{\gamma^2 m'^2}$ elements, we have with probability at least $1 - n^{-10}$ that

$$(1 - \gamma) (m' \cdot s)^2 \cdot \frac{s'}{s^2} \leq |\mathbf{p} \cap \mathbf{z}|.$$

A similar result gives the upper bound on $|\mathbf{p} \cap \mathbf{z}|$ for $s' \geq \frac{30 \ln(n)}{\gamma^2 m^2}$. As $m' \leq m$, we maximize s' by $\frac{30 \ln(n)}{\gamma^2 m'^2}$ and thus ensure both bounds with probability at least $1 - 2n^{-10}$ using a union bound. ◀

We are generally going to use γ as the same fixed parameter (to be determined later) every time we invoke the sampling of Lemma 5.

Lemma 5 will be used to show that sampling after squaring will not distort the Jaccard similarity of a pair of vectors too much, and hence we get the benefits of squaring without the exploding vector dimensions. We start by bounding the resulting sizes for each of $|\mathbf{a}|$, $|\mathbf{b}|$ and $|\mathbf{a} \cap \mathbf{b}|$ for any choice of $\mathbf{a}, \mathbf{b} \in \mathcal{A} \times \mathcal{B}$ from squaring and sampling i times.

► **Lemma 6.** *Let \mathbf{v} be a set from a universe of size d or the intersection of such two sets. Let $\mathbf{f}(\mathbf{v}, \mathbf{i})$ denote the resulting set after running i iterations of the squaring-and-sampling reduction on set \mathbf{v} for $i \geq 1$. We have*

$$(1 - \gamma)^{2^i} \frac{|\mathbf{v}|^{2^i}}{d^{2^i}} s_i \leq |\mathbf{f}(\mathbf{v}, \mathbf{i})| \leq (1 + \gamma)^{2^i} \frac{|\mathbf{v}|^{2^i}}{d^{2^i}} s_i$$

with probability at least $1 - 2in^{-10}$ where $s_i \geq \frac{30 \ln(n)d^{2^i}}{\gamma^2(1-\gamma)^{2^i-2}|\mathbf{v}|^{2^i}}$.

Proof. Let \mathbf{v} be as described. We show the lemma by induction on i . Clearly, when squaring the vector \mathbf{v} once, i.e., for $i = 1$, the resulting vector has Hamming weight $|\mathbf{v}|^2$ and dimension d^2 . Hence, by Lemma 5 we have

$$(1 - \gamma) \frac{|\mathbf{v}|^2}{d^2} \cdot s_1 \leq |\mathbf{f}(\mathbf{v}, \mathbf{1})| \leq (1 + \gamma) \frac{|\mathbf{v}|^2}{d^2} \cdot s_1$$

with probability at least $1 - 2n^{-10}$ for our choice of s_1 . Assume now that after $i - 1$ iterations the following bounds hold:

$$(1 - \gamma)^{2^{i-1}-1} \frac{|\mathbf{v}|^{2^{i-1}}}{d^{2^{i-1}}} s_{i-1} \leq |\mathbf{f}(\mathbf{v}, \mathbf{i} - \mathbf{1})| \leq (1 + \gamma)^{2^{i-1}-1} \frac{|\mathbf{v}|^{2^{i-1}}}{d^{2^{i-1}}} s_{i-1}. \quad (3)$$

Then Lemma 5 gives that after i iterations of the squaring-and-sampling reduction, we have

$$(1 - \gamma)^{2^i-1} \frac{|\mathbf{v}|^{2^i} s_{i-1}^2}{d^{2^i}} \cdot \frac{s_i}{s_{i-1}^2} \leq |\mathbf{f}(\mathbf{v}, \mathbf{i})| \leq (1 + \gamma)^{2^i-1} \frac{|\mathbf{v}|^{2^i} s_{i-1}^2}{d^{2^i}} \cdot \frac{s_i}{s_{i-1}^2}$$

with probability at least $1 - 2n^{-10}$ for $s_i \geq \frac{30 \ln(n)d^{2^i}}{\gamma^2(1-\gamma)^{2^i-2}|\mathbf{v}|^{2^i}}$. This particularly means that

$$(1 - \gamma)^{2^i} \frac{|\mathbf{v}|^{2^i}}{d^{2^i}} \cdot s_i \leq |\mathbf{f}(\mathbf{v}, \mathbf{i})| \leq (1 + \gamma)^{2^i} \frac{|\mathbf{v}|^{2^i}}{d^{2^i}} \cdot s_i.$$

Now, to ensure these bounds, we assumed that $|\mathbf{f}(\mathbf{v}, \mathbf{i} - \mathbf{1})|$ satisfies certain bounds (see (3)). So in order to ensure that $\mathbf{f}(\mathbf{v}, \mathbf{i})$ satisfies the given bounds, we need $\mathbf{f}(\mathbf{v}, \mathbf{j})$ to satisfy similar bounds for every $1 \leq j \leq i$. By a union bound, we see that $|\mathbf{f}(\mathbf{v}, \mathbf{j})|$ satisfies both upper and lower bounds for all $1 \leq j \leq i$ (simultaneously) with probability at least $1 - 2in^{-10}$ when sampling $s_j \geq \frac{30 \ln(n)d^{2^j}}{\gamma^2(1-\gamma)^{2^j-2}|\mathbf{v}|^{2^j}}$ at step j . Hence, $|\mathbf{f}(\mathbf{v}, \mathbf{i})|$ satisfies the given bound with probability at least $1 - 2in^{-10}$. ◀

The next section will use Lemma 6 to bound the Jaccard similarity after i iterations of the squaring/sampling reduction.

4.2 Combining the bounds

For a given pair of vectors \mathbf{a} and \mathbf{b} , Lemma 6 gives upper and lower bounds on the Jaccard similarity $J = J(\mathbf{f}(\mathbf{a}, \mathbf{i}), \mathbf{f}(\mathbf{b}, \mathbf{i}))$. We claim that with probability at least $1 - 6in^{-10}$:

$$\begin{aligned} J &\geq \frac{(1 - \gamma)^{2^i-1} \frac{|\mathbf{a} \cap \mathbf{b}|^{2^i}}{d^{2^i}} s_i}{(1 + \gamma)^{2^i-1} \frac{|\mathbf{a}|^{2^i}}{d^{2^i}} s_i + (1 + \gamma)^{2^i-1} \frac{|\mathbf{b}|^{2^i}}{d^{2^i}} s_i - (1 - \gamma)^{2^i-1} \frac{|\mathbf{a} \cap \mathbf{b}|^{2^i}}{d^{2^i}} s_i} \\ &\geq \frac{(1 - \gamma)^{2^i} |\mathbf{a} \cap \mathbf{b}|^{2^i}}{(1 + \gamma)^{2^i} (|\mathbf{a}|^{2^i} + |\mathbf{b}|^{2^i}) - (1 - \gamma)^{2^i} |\mathbf{a} \cap \mathbf{b}|^{2^i}} \\ J &\leq \frac{(1 + \gamma)^{2^i-1} \frac{|\mathbf{a} \cap \mathbf{b}|^{2^i}}{d^{2^i}} s_i}{(1 - \gamma)^{2^i-1} \frac{|\mathbf{a}|^{2^i}}{d^{2^i}} s_i + (1 - \gamma)^{2^i-1} \frac{|\mathbf{b}|^{2^i}}{d^{2^i}} s_i - (1 + \gamma)^{2^i-1} \frac{|\mathbf{a} \cap \mathbf{b}|^{2^i}}{d^{2^i}} s_i} \\ &\leq \frac{(1 + \gamma)^{2^i} |\mathbf{a} \cap \mathbf{b}|^{2^i}}{(1 - \gamma)^{2^i} (|\mathbf{a}|^{2^i} + |\mathbf{b}|^{2^i}) - (1 + \gamma)^{2^i} |\mathbf{a} \cap \mathbf{b}|^{2^i}}. \end{aligned}$$

This is easily seen by taking a union bound over the probabilities that each of $|\mathbf{a}|$, $|\mathbf{b}|$ and $|\mathbf{a} \cap \mathbf{b}|$ violate either the upper or the lower bound. Next, we claim that these bounds imply:

$$\begin{aligned} J &\geq \frac{(1 - \gamma)^{2^i} |\mathbf{a} \cap \mathbf{b}|^{2^i}}{(1 + 4\gamma)^{2^i} (|\mathbf{a}|^{2^i} + |\mathbf{b}|^{2^i} - |\mathbf{a} \cap \mathbf{b}|^{2^i})} \geq \frac{(1 - \gamma)^{2^i} |\mathbf{a} \cap \mathbf{b}|^{2^i}}{(1 + \gamma)^{2^i} (|\mathbf{a}|^{2^i} + |\mathbf{b}|^{2^i}) - (1 - \gamma)^{2^i} |\mathbf{a} \cap \mathbf{b}|^{2^i}} \\ J &\leq \frac{(1 + \gamma)^{2^i} |\mathbf{a} \cap \mathbf{b}|^{2^i}}{(1 - \gamma)^{2^i} (|\mathbf{a}|^{2^i} + |\mathbf{b}|^{2^i}) - (1 + \gamma)^{2^i} |\mathbf{a} \cap \mathbf{b}|^{2^i}} \leq \frac{(1 + \gamma)^{2^i} |\mathbf{a} \cap \mathbf{b}|^{2^i}}{(1 - 4\gamma)^{2^i} (|\mathbf{a}|^{2^i} + |\mathbf{b}|^{2^i} - |\mathbf{a} \cap \mathbf{b}|^{2^i})}. \end{aligned}$$

The argument can be found in Appendix A in the full version on ArXiv [8, App. A]. In particular, we have argued for the following lemma. We ignore the sample size for now and discuss it in Section 4.3.

► **Lemma 7.** *Let \mathcal{A} and \mathcal{B} be an instance of Bichromatic Closest Pair with Jaccard similarity. After applying the Squaring and Sampling mapping, f , i times as previously described to each set in \mathcal{A} and \mathcal{B} , we have for all n^2 pairs $(\mathbf{a}, \mathbf{b}) \in \mathcal{A} \times \mathcal{B}$ in the instance that:*

$$\frac{\left(\frac{1-\gamma}{1+4\gamma}\right)^{2^i} |\mathbf{a} \cap \mathbf{b}|^{2^i}}{|\mathbf{a}|^{2^i} + |\mathbf{b}|^{2^i} - |\mathbf{a} \cap \mathbf{b}|^{2^i}} \leq J(\mathbf{f}(\mathbf{a}, \mathbf{i}), \mathbf{f}(\mathbf{b}, \mathbf{i})) \leq \frac{\left(\frac{1+\gamma}{1-4\gamma}\right)^{2^i} |\mathbf{a} \cap \mathbf{b}|^{2^i}}{|\mathbf{a}|^{2^i} + |\mathbf{b}|^{2^i} - |\mathbf{a} \cap \mathbf{b}|^{2^i}}$$

with probability at least $1 - 6in^{-8}$.

Hence, with high probability none of the Jaccard similarities diverge too much from (2) due to sampling. This was exactly what we wanted, as this allows us to reduce the dimension by sampling.

4.3 Summing up

Recall that in our setting we reduce from instances where the set sizes of all red and blue sets are fixed. We now describe thresholds such that solving the instances constructed by the reduction f cannot be done in truly subquadratic time.

► **Lemma 8.** *Let \mathcal{A} and \mathcal{B} be two collections of n sets from a universe of dimension d , where all sets in \mathcal{A} have size y and all sets in \mathcal{B} have size z . Assume that $(\mathcal{A}, \mathcal{B})$ is taken from a family of instances of Bichromatic Closest Pair with Jaccard similarity, which require time $\Omega(n^{2-\delta})$ for thresholds $t_1 = \frac{x_1}{y+z-x_1}$ and $t_2 = \frac{x_2}{y+z-x_2}$. The reduction which applies f i times to each set in $\mathbf{s} \in \mathcal{A} \cup \mathcal{B}$ for $i \geq 1$ constructs an instance of Bichromatic Closest Pair with Jaccard similarity, which requires time $\Omega(n^{2-\delta})$ time for thresholds*

$$t'_1 = \left(\frac{1-\gamma}{1+4\gamma}\right)^{2^i} \frac{x_1^{2^i}}{y^{2^i} + z^{2^i} - x_1^{2^i}}, \quad \text{and} \quad t'_2 = \left(\frac{1+\gamma}{1-4\gamma}\right)^{2^i} \frac{x_2^{2^i}}{y^{2^i} + z^{2^i} - x_2^{2^i}}.$$

whose solution provides a valid solution to the original instance with high probability when sampling $s_j > \frac{30 \ln(n) d^{2^j}}{\gamma^2 (1-\gamma)^{2^j-2} x_2^{2^j}}$ at each step $1 \leq j \leq i$.

Proof. Lemma 7 ensures that with high probability a solution to the constructed instance provides a valid solution to the original instance, since no pair of sets is likely to have Jaccard similarities that deviate beyond the chosen thresholds.

In Lemma 7 we skipped the discussion of the sample size at each iteration – we will argue for it now. From Lemma 6, it is easily seen that we maximize the needed sample size for all of $|\mathbf{a}|$, $|\mathbf{b}|$ or $|\mathbf{a} \cap \mathbf{b}|$ for any choice of \mathbf{a} and \mathbf{b} in iteration i by

$$s_i > \frac{30 \ln(n) d^{2^i}}{\gamma^2 (1-\gamma)^{2^i-2} \min_{(\mathbf{a}, \mathbf{b}) \in \mathcal{A} \times \mathcal{B}} \{|\mathbf{a} \cap \mathbf{b}|\}^{2^i}}.$$

Hence, sampling s_i elements from the universe will ensure that each of the upper and lower bounds for either $|\mathbf{a}|$, $|\mathbf{b}|$ or $|\mathbf{a} \cap \mathbf{b}|$ will fail with probability at most n^{-10} in that iteration. As $\min_{(\mathbf{a}, \mathbf{b}) \in \mathcal{A} \times \mathcal{B}} \{|\mathbf{a} \cap \mathbf{b}|\}$ is unknown, we instead use x_2 , which was the intersection size for a pair with Jaccard similarity j_2 . Such a pair need not exist, but as the set sizes are fixed, x_2 can be easily computed.

We have left to argue that the pairs with intersection smaller than x_2 also satisfy the bounds in Lemma 7 with high probability. The main observation is that they only need to satisfy the upper bound, as the resulting Jaccard similarities need only to stay below the lower threshold, t'_2 – the Jaccard similarities can become arbitrarily small without affecting the result.

By bounding the size of each term as we did in Lemma 6 using the chosen s_i , we see that the error probabilities are still at most n^{-10} for each of $|\mathbf{a}|$, $|\mathbf{b}|$ and $|\mathbf{a} \cap \mathbf{b}|$ for any choice of $(\mathbf{a}, \mathbf{b}) \in \mathcal{A} \times \mathcal{B}$. ◀

5 Main Result

We are now ready to prove Theorem 2. We first give some intuition behind the proof and state a few lemmas to ease the proof. For convenience we restate Theorem 2.

► **Theorem 9.** *Assuming the Orthogonal Vectors Conjecture (OVC), the following holds: for any $\delta > 0$, there exists an $\varepsilon > 0$ such that for any given $j_2 < j_1 < 1 - \delta$ satisfying $j_1 \leq j_2^{1-\varepsilon}$, solving Bichromatic Closest Pair with Jaccard similarity for n red and n blue sets for sets from a universe of size $\ln(n)/j_2^{O(\log(1/j_1))}$ for thresholds j_1 and j_2 requires time $\Omega(n^{2-\delta})$.*

5.1 Intuition

The proof of Theorem 2 reduces instances of Bichromatic Closest Pair as described in Section 2.3 by composing three reductions, that together construct instances of Bichromatic Closest Pair with Jaccard similarity, which requires time $\Omega(n^{2-\delta})$ for the given thresholds j_1 and j_2 and some ε . A short description of each of the reductions can be found in Section 3. Below, we give three lemmas showing that these reductions preserve hardness.

The first lemma states that adding common elements to all sets in the instance will preserve hardness. This reduction increases the Jaccard similarity of all pairs of red and blue sets, and by choice of the number of added elements, we ensure that pairs of sets that initially had Jaccard similarity higher than the *lower* threshold will get Jaccard similarity greater than $1 - \delta$. Hence, we get hardness for thresholds that are greater than $1 - \delta$. From this point we can decrease the thresholds using two other reductions to achieve the given thresholds, that by assumption are less than $1 - \delta$.

The second lemma states that the squaring-and-sampling reduction, discussed in detail in Section 4, preserves hardness. The squaring-and-sampling reduction allows us to decrease the thresholds, so they come close to j_1 and j_2 . Finally, the third lemma states that the reduction, which adds elements to only red sets will still preserve hardness. This reduction ensures that we can decrease the Jaccard similarity further. We will use it in such a way, that we effectively multiply the upper bound by a well-chosen α that ensures that the upper threshold is j_1 after this reduction. The proof ends by picking an ε , such that j_2 is strictly greater than the current lower threshold, and thus preserves hardness for the thresholds j_1 and j_2 .

5.2 Supporting Lemmas

In the following, assume that \mathcal{A} and \mathcal{B} are collections of n red and n blue sets from a universe U , respectively.

► **Lemma 9.** *Let $0 < \delta \leq 1$ be given and let $(\mathcal{A}, \mathcal{B})$ be any instance of Bichromatic Closest Pair with Jaccard similarity as described in Lemma 3. Define $\ell := \max_{\mathbf{q} \in \mathcal{A} \cup \mathcal{B}} \{|\mathbf{q}|\} \cdot (1/\delta - 1)$ and $\mathbf{x} := \{x_1, \dots, x_\ell\}$ such that $\mathbf{x} \cap (\mathcal{A} \cup \mathcal{B}) = \emptyset$, and further define the mapping $g : \mathcal{A} \cup \mathcal{B} \rightarrow \mathcal{A}' \cup \mathcal{B}'$ by $g(\mathbf{v}) = \mathbf{v} \cup \mathbf{x}$ where $\mathcal{A}' = \mathcal{A} \cup \mathbf{x}$ and equivalently $\mathcal{B}' = \mathcal{B} \cup \mathbf{x}$. The reduction that applies g to every element of \mathcal{A} and \mathcal{B} generates an instance $(\mathcal{A}', \mathcal{B}')$ of Bichromatic Closest Pair with Jaccard similarity that requires time $\Omega(n^{2-\delta})$ for some thresholds $t'_1, t'_2 \geq 1 - \delta$.*

Proof. First, note that if $\mathbf{v} \in \mathcal{A}$, then $g(\mathbf{v}) \in \mathcal{A}'$ and similarly if $\mathbf{v} \in \mathcal{B}$ then $g(\mathbf{v}) \in \mathcal{B}'$. We recall that instances of Bichromatic Closest Pair as described in Lemma 3 are constructed such that all red sets have the same size and all blue sets have the same size. We also have $\max_{\mathbf{q} \in \mathcal{A} \cup \mathcal{B}} \{|\mathbf{q}|\} = |\mathbf{a}|$, for any $\mathbf{a} \in \mathcal{A}$, since the sets in \mathcal{A} were larger than the sets in \mathcal{B} . It is easy to see that hardness is preserved under the reduction.

74:10 Hardness of Bichromatic Closest Pair with Jaccard Similarity

We finally argue that the resulting thresholds are larger than $1 - \delta$: Let (\mathbf{a}, \mathbf{b}) be any pair from $\mathcal{A} \times \mathcal{B}$ which has Jaccard similarity at least t_2 and let $\mathbf{a}' = g(\mathbf{a})$ and $\mathbf{b}' = g(\mathbf{b})$. We argue that any such pair satisfies $|\mathbf{a} \cap \mathbf{b}| \geq \frac{|\mathbf{b}|}{2}$: Note that with these particular instances of Bichromatic Closest Pair and from the proof of Lemma 4, we have

$$J(\mathbf{a}, \mathbf{b}) = \frac{|\mathbf{a} \cap \mathbf{b}|}{|\mathbf{a} \cup \mathbf{b}|} = \frac{|\mathbf{a} \cap \mathbf{b}|}{Tm + m - |\mathbf{a} \cap \mathbf{b}|} \geq t_2 = \frac{t_1}{2} = \frac{1/T}{2}.$$

Since $|\mathbf{b}| = m \geq |\mathbf{a} \cap \mathbf{b}|$, this implies

$$|\mathbf{a} \cap \mathbf{b}| \geq \frac{m}{2} + \frac{m}{2T} - \frac{|\mathbf{a} \cap \mathbf{b}|}{2T} \Rightarrow |\mathbf{a} \cap \mathbf{b}| \geq m/2 = |\mathbf{b}|/2.$$

We will consider the Jaccard similarity of \mathbf{a}' and \mathbf{b}' :

$$\begin{aligned} J(\mathbf{a}', \mathbf{b}') &= \frac{|\mathbf{a} \cap \mathbf{b}| + |\mathbf{a}|(1/\delta - 1)}{(|\mathbf{a}| + |\mathbf{a}|(1/\delta - 1)) + (|\mathbf{b}| + |\mathbf{a}|(1/\delta - 1)) - (|\mathbf{a} \cap \mathbf{b}| + |\mathbf{a}|(1/\delta - 1))} \\ &= \frac{|\mathbf{a} \cap \mathbf{b}| + |\mathbf{a}|(1/\delta - 1)}{|\mathbf{a}|/\delta + |\mathbf{b}| - |\mathbf{a} \cap \mathbf{b}|}. \end{aligned}$$

By assumption $|\mathbf{a} \cap \mathbf{b}| \geq \frac{|\mathbf{b}|}{2}$, so:

$$\begin{aligned} \frac{|\mathbf{a} \cap \mathbf{b}| + |\mathbf{a}|(1/\delta - 1)}{|\mathbf{a}|/\delta + |\mathbf{b}| - |\mathbf{a} \cap \mathbf{b}|} &\geq \frac{|\mathbf{b}|/2 + |\mathbf{a}|(1/\delta - 1)}{|\mathbf{a}|/\delta + |\mathbf{b}|/2} \geq 1 - \delta \\ \Leftrightarrow \frac{|\mathbf{b}|}{2} + |\mathbf{a}|(1/\delta - 1) &\geq |\mathbf{a}|(1/\delta - 1) + \frac{|\mathbf{b}|}{2} - \frac{|\mathbf{b}|\delta}{2} \end{aligned}$$

which is always satisfied. Hence, $J(\mathbf{a}', \mathbf{b}') \geq 1 - \delta$ for any choice of $\delta > 0$, and so, we construct an instance where every pair with Jaccard similarity higher than t_2 will have Jaccard similarity higher than $1 - \delta$. Thus, there are thresholds that are greater than $1 - \delta$, that make the constructed instance hard. \blacktriangleleft

► **Lemma 10.** *Let $0 < \delta \leq 1$ be given and consider any instance of Bichromatic Closest Pair with Jaccard similarity, $(\mathcal{A}, \mathcal{B})$, from a family of instances which require time $\Omega(n^{2-\delta})$ for thresholds t_1 and t_2 . Using the reduction f defined in Section 4 on each $\mathbf{v} \in \mathcal{A} \cup \mathcal{B}$ for i iterations where $i \geq 1$, we construct a valid instance of Bichromatic Closest Pair with Jaccard similarity with high probability, which requires time $\Omega(n^{2-\delta})$ for thresholds that are decreasing functions of i .*

Proof. The lemma follows immediately from Lemma 8. \blacktriangleleft

► **Lemma 11.** *Let $0 < \delta \leq 1$ be given and consider any instance of Bichromatic Closest Pair with Jaccard similarity, $(\mathcal{A}, \mathcal{B})$, from a family of instances which require time $\Omega(n^{2-\delta})$ for thresholds t_1 and t_2 . Define $\ell := \max_{\mathbf{q} \in \mathcal{A} \cup \mathcal{B}} \{|\mathbf{q}|\} \cdot (1/\alpha - 1)$ and $\mathbf{y} := \{y_1, \dots, y_\ell\}$ such that $\mathbf{y} \cap (\mathcal{A} \cup \mathcal{B}) = \emptyset$. Define mapping $h : \mathcal{A} \rightarrow \mathcal{A}'$ where $\mathcal{A}' = \mathcal{A} \cup \mathcal{Y}$ by $h(\mathbf{a}) = \mathbf{a} \cup \mathbf{y}$. The reduction that applies h to every element of \mathcal{A} generates an instance $(\mathcal{A}', \mathcal{B})$ of Bichromatic Closest Pair with Jaccard similarity that requires time $\Omega(n^{2-\delta})$ for some thresholds t'_1, t'_2 .*

Proof. Clearly, hardness is preserved under the reduction that simply adds new elements to all red sets. In particular this reduction decreases the thresholds by decreasing the similarity between red and blue pairs. \blacktriangleleft

5.3 Proof outline for Theorem 2

Proof. For simplicity and readability we leave out most of the calculations – details can be found in Appendix B in the full version on ArXiv [8, App. B].

Let $\delta > 0$ be given and let j_1, j_2 be given such that $j_2 < j_1 < 1 - \delta$. Take any instance of Bichromatic Closest Pair with Jaccard similarity satisfying the properties described in Lemma 3. Recall from this lemma that $T = O\left(\frac{1}{\varepsilon}\right)$.

Apply the reductions from first Lemma 9 to achieve an instance, which requires time $\Omega(n^{2-\delta})$ for thresholds greater than $1 - \delta$. We wish to reduce to an instance that is hard for smaller thresholds j_1 and j_2 . The reduction from Lemma 10 is used to decrease the thresholds, where we pick the largest i , such that the resulting upper threshold t_1 is no smaller than j_1 , i.e., $t_1 \geq j_1$. This reduction decreases the thresholds until the upper threshold is only slightly greater than j_1 . Now, let $\alpha = \frac{j_1}{t_1}$ and apply the reduction from Lemma 11 to ensure that the resulting upper threshold is now equal to j_1 . This eventually gives an instance of Bichromatic Closest Pair with Jaccard similarity, which cannot be solved in time $O(n^{2-\delta})$ for thresholds

$$t'_1 = \alpha \left(\frac{1-\gamma}{1+4\gamma} \right)^{2^i} \left(\frac{\delta}{T} + 1 - \delta \right)^{2^i}$$

$$t'_2 = \left(\frac{1+\gamma}{1-4\gamma} \right)^{2^i} \frac{\left(\frac{\delta}{2T} + 1 - \delta \right)^{2^i}}{\frac{1}{\alpha} + \left(\frac{\delta}{T} + 1 - \delta \right)^{2^i} - \left(\frac{\delta}{2T} + 1 - \delta \right)^{2^i}}$$

where we observe that by construction $t'_1 = \alpha \cdot t_1 = j_1$. We refer to Appendix B in the full version on ArXiv for the calculations [8, App. B]. So we have constructed an instance which is hard for thresholds j_1 and t'_2 .

Set $t_2^* = \left(\frac{1+\gamma}{1-4\gamma} \right)^{2^i} \left(\frac{\delta}{2T} + 1 - \delta \right)^{2^i}$. Then $t'_2 < \alpha t_2^*$ and so the hardness for $t'_1 = j_1$ and t'_2 implies hardness for $t'_1 = j_1$ and αt_2^* . We show that there is an ε that only depends on δ such that $\alpha t_2^* < j_2$. Then the hardness for $t'_1 = j_1$ and αt_2^* implies hardness for the given j_1 and j_2 .

Note that we have chosen $\alpha \geq t_1$, since otherwise i could not be maximal. So we have:

$$\frac{\log(j_1)}{\log(\alpha t_2^*)} = \frac{\log(\alpha t_1)}{\log(\alpha t_2^*)} \leq \frac{\log(t_1^2)}{\log\left(t_1 \cdot \left(\frac{1+\gamma}{1-4\gamma}\right)^{2^i} \cdot \left(\frac{\delta}{2T} + 1 - \delta\right)^{2^i}\right)}$$

$$= \frac{2^i \cdot \log\left(\left(\frac{1-\gamma}{1+4\gamma}\right)^2 \cdot (\delta/T + 1 - \delta)^2\right)}{2^i \cdot \log\left(\left(\frac{1-\gamma}{1+4\gamma}\right) \cdot (\delta/T + 1 - \delta) \left(\frac{1+\gamma}{1-4\gamma}\right) \cdot \left(\frac{\delta}{2T} + 1 - \delta\right)\right)}.$$

We need to show that this expression is bounded by $1 - \varepsilon$ for some ε that depends on δ , but not on j_1 and j_2 . Observe that the factors 2^i cancel out and we may pick γ small enough that it can essentially be ignored. We show in Appendix B in the full version on ArXiv [8, App. B] that we can use any $\gamma < \min\left\{\frac{1}{2^{i+1}}, \frac{\delta}{20T}\right\}$. Then for given δ , there exists an ε such that the expression is bounded by $1 - \varepsilon$, since T can be considered a constant for a fixed δ . Recall that T was defined in Lemma 3. By the assumption $j_1 \leq j_2^{1-\varepsilon}$ we then have $\alpha t_2^* < j_2$. Then the hardness of t'_1 and αt_2^* where $t'_1 = j_1$ and $\alpha t_2^* < j_2$, implies the desired hardness for the given j_1 and j_2 .

We finally argue about the size of the universe of the instance constructed by the compositions of reductions described. In the following, d is the size of the universe of the initial instance of Bichromatic Closest Pair with Jaccard instance. In the proof of Lemma 8, we argued that we could use x_2 , which was the size of the intersection for a pair with Jaccard similarity j_2 , in the sample size s_i , which means that

$$\begin{aligned} s_i &\geq \frac{30 \ln(n) d^{2^i}}{\gamma^2 (1 - \gamma)^{2^i} x_2^{2^i}} = \frac{30 \ln(n) d^{2^i}}{\gamma^2 (1 - \gamma)^{2^i} (j_2 (|a| + |b| - x_2))^{2^i}} \\ &= \frac{30 \ln(n)}{\gamma^2 (1 - \gamma)^{2^i} j_2^{2^i}} \cdot \left(\frac{\delta + 1}{1 + \frac{\delta}{2T}} \right)^{2^i}. \end{aligned}$$

Again, the calculations can be found in Appendix B in the full version on ArXiv [8, App. B]. Hence, the sets constructed by the composition of reductions come from a universe whose size is bounded by

$$|U| \leq s_i + s_i(1/\alpha - 1) = \frac{s_i}{\alpha} \leq \frac{30 \ln(n)}{\gamma^2 j_2^{2^i}} \left(\frac{\delta + 1}{(\frac{\delta}{2T} + 1 - \delta)(\frac{\delta}{2T} + 1)} \right)^{2^i} \left(\frac{1 + 4\gamma}{(1 - \gamma)^2} \right)^{2^i}$$

By Assumption $t_1'^2 < j_1 \leq t_1'$, which implies that $2^i = O\left(\frac{\log j_1}{\log c}\right) = O\left(\log \frac{1}{j_1}\right)$ for constant $c < 1$. Hence, we conclude that the size of the universe is $\ln(n)/j_2^{O(\log 1/j_1)}$. This finishes the proof of Theorem 2. ◀

6 Final Comments

On a final note, we remark that one can obtain a result similar to Theorem 2 for Braun-Blanquet similarity. Recall that we define Braun-Blanquet similarity for a pair of sets $(\mathbf{a}, \mathbf{b}) \in \mathcal{A} \times \mathcal{B}$ as

$$BB(\mathbf{a}, \mathbf{b}) = \frac{|\mathbf{a} \cap \mathbf{b}|}{\max\{|\mathbf{a}|, |\mathbf{b}|\}} \in [0, 1]$$

In fact, the proof is slightly simpler than the one given in Section 5.3 and the calculations are somewhat nicer. The proof ideas, i.e., the choice and order of reductions, are exactly the same and should be easy to carry out by following the structure of the proof of Theorem 2.

The main open problem we leave is whether existing upper bounds are near-optimal when ε is an arbitrary constant between 0 and 1. Our techniques only work when ε is sufficiently small.

References

- 1 Andrei Z Broder. On the resemblance and containment of documents. In *Compression and complexity of sequences 1997. proceedings*, pages 21–29. IEEE, 1997.
- 2 Lijie Chen. On The Hardness of Approximate and Exact (Bichromatic) Maximum Inner Product. In *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, pages 14:1–14:45, 2018. doi:10.4230/LIPIcs.CCC.2018.14.
- 3 Lijie Chen and Ryan Williams. An Equivalence Class for Orthogonal Vectors. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 21–40, 2019. doi:10.1137/1.9781611975482.2.

- 4 Tobias Christiani and Rasmus Pagh. Set similarity search beyond MinHash. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1094–1107, 2017. doi:10.1145/3055399.3055443.
- 5 Ashish Goel, Aneesh Sharma, Dong Wang, and Zhijun Yin. Discovering similar users on twitter. In *11th Workshop on Mining and Learning with Graphs*, 2013.
- 6 Pankaj Gupta, Ashish Goel, Jimmy J. Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. WTF: the who to follow service at twitter. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pages 505–514, 2013. doi:10.1145/2488388.2488433.
- 7 Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 604–613, 1998. doi:10.1145/276698.276876.
- 8 Rasmus Pagh, Nina Stausholm, and Mikkel Thorup. Hardness of Bichromatic Closest Pair with Jaccard Similarity, 2019. arXiv:1907.02251.
- 9 Aviad Rubinfeld. Hardness of approximate nearest neighbor search. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1260–1268, 2018. doi:10.1145/3188745.3188916.
- 10 Gregory Valiant. Finding Correlations in Subquadratic Time, with Applications to Learning Parities and the Closest Pair Problem. *J. ACM*, 62(2):13:1–13:45, 2015. doi:10.1145/2728167.
- 11 Virginia Vassilevska Williams. Some Open Problems in Fine-Grained Complexity. *SIGACT News*, 49(4):29–35, 2018. doi:10.1145/3300150.3300158.

Compact Oblivious Routing

Harald Räcke

Department of Informatics, TU München, Germany
raecke@in.tum.de

Stefan Schmid

Faculty of Computer Science, University of Vienna, Austria
stefan_schmid@univie.ac.at

Abstract

Oblivious routing is an attractive paradigm for large distributed systems in which centralized control and frequent reconfigurations are infeasible or undesired (e.g., costly). Over the last almost 20 years, much progress has been made in devising oblivious routing schemes that guarantee close to optimal load and also algorithms for constructing such schemes efficiently have been designed. However, a common drawback of existing oblivious routing schemes is that they are not compact: they require large routing tables (of polynomial size), which does not scale.

This paper presents the first oblivious routing scheme which guarantees close to optimal load and is compact at the same time – requiring routing tables of *polylogarithmic* size. Our algorithm maintains the polylogarithmic competitive ratio of existing algorithms, and is hence particularly well-suited for emerging large-scale networks.

2012 ACM Subject Classification Networks → Routing protocols; Theory of computation → Routing and network design problems; Networks → Network algorithms

Keywords and phrases Oblivious Routing, Compact Routing, Competitive Analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.75

1 Introduction

1.1 Motivation

With the increasing scale and dynamics of large networked systems, observing and reacting to changes in the workload and reconfiguring the routing accordingly becomes more and more difficult. Not only does a larger network and more dynamic workload require more fine-grained monitoring and control (which both introduce overheads), also the process of re-routing traffic itself (see e.g. [15]) can lead to temporary performance degradation and transient inconsistencies.

Oblivious routing provides an attractive alternative which avoids these reconfiguration overheads while being *competitive*, i.e., while guaranteeing route allocations which are almost as good as adaptive solutions. It is hence not surprising that oblivious routing has received much attention over the last two decades. Indeed, today, we have a good understanding of fast (i.e., polynomial-time constructable) and “competitive” oblivious routing algorithms (achieving a polylogarithmic approximation of the load, which is optimal).

However, while oblivious routing seems to be the perfect paradigm for emerging large networked systems, there is a fly in the ointment. Oblivious routing algorithms that aim to minimize congestion require large routing tables: namely *polynomial* in the network size. This is problematic, as fast memory in routers is expensive, not only in terms of monetary costs but also in terms of power consumption.

The goal of this paper is to design oblivious routing schemes which only require small routing tables (which are *compact*), and that at the same time still guarantee a close-to-optimal load.



© Harald Räcke and Stefan Schmid;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 75; pp. 75:1–75:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.2 The Problem in a Nutshell

The network is given as an undirected graph $G = (V, E)$ with n vertices. The edges E are weighted by a capacity function $\text{cap} : V \times V \rightarrow \mathbb{R}_0^+$; if $\{x, y\} \in E$, the function returns 0, otherwise a positive value.

The *oblivious routing problem* is to set up a unit flow for each source-target pair $(s, t) \in V \times V$ that determines how demand between s and t is routed in the network G . This unit flow is pre-specified without knowing the actual demands. When a demand vector \vec{d} is given that specifies for each pair of vertices the amount of traffic to be sent, the demand-vector is routed by simply scaling the unit flow between a pair (s, t) by the corresponding demand d_{st} between the two vertices. This means that traffic is routed along *pre-computed paths* and that no path-selection is done dynamically.

The congestion $C_{\text{obl}}(G, \vec{d})$ resulting from a given oblivious routing scheme, is then compared to the optimal possible congestion $C_{\text{opt}}(G, \vec{d})$ that can be obtained for demand vector \vec{d} in G . The *competitive ratio* of the oblivious routing scheme is defined as

$$\max_{\vec{d}} \frac{C_{\text{obl}}(G, \vec{d})}{C_{\text{opt}}(G, \vec{d})}$$

In this paper, we are interested in designing packet forwarding rules that allow the packets to follow a flow of an oblivious routing scheme with a good competitive ratio. Apart from the competitiveness of the underlying oblivious routing scheme one goal is to encode the forwarding rules *compactly* with small space requirement. In particular we require that for a vertex v the space requirement is only $O(\deg(v) \text{polylog}(n))$, where $\deg(v)$ is the degree of the vertex. In other words, this means we roughly require a polylogarithmic number of rules *per network link*. It seems unavoidable to let the memory requirement of a vertex depend on its degree as otherwise the routing scheme might not be able to efficiently utilize all network links.

In addition to the competitive ratio, the runtime, and the table size, we are also interested in the required vertex labels (i.e., their size) and the required packet header size.

1.3 Our Contributions

This paper presents the first *compact* oblivious routing scheme. Our approach builds upon an oblivious path selection scheme based on classic decomposition trees, which is then adapted to improve scalability, and in particular, to ensure small routing tables and message headers, while preserving polynomial runtime (for constructing the routing tables) and a polylogarithmic competitive ratio.

We present two different implementations of our approach and our results come in two different flavors accordingly (more detailed theorems will follow):

► **Theorem 1.** *There exist oblivious routing strategies that achieve a polylogarithmic competitive ratio w.r.t. the congestion and require routing tables of polylogarithmic size for*

1. *networks with arbitrary edge capacities which have a decomposition tree of bounded degree, and for*
2. *arbitrary networks with uniform edge capacities.*

Our algorithms only require small (polylogarithmic) header sizes and vertex labels. The routing tables can be constructed in polynomial time.

Networks for which there are decomposition trees of small degree include for example (constant-degree) grids. The exact requirements that a decomposition tree has to fulfill will be given later.

2 Algorithm and Analysis

This section describes an oblivious path selection scheme for general undirected networks that obtains close to optimal congestion and can be implemented with routing tables and routing headers of small size. In a nutshell, our algorithm leverages a path selection scheme for general networks that guarantees a good competitive ratio w.r.t. congestion, and then adapts it so that it can be implemented with small space requirements. We discuss the two phases of this algorithm in turn.

2.1 Oblivious Path Selection Scheme

There exist essentially two path selection schemes that could be used as a basis for our approach. First, there is the original result by Räcke [32] who showed that oblivious routing with a polylogarithmic competitive ratio is possible in general networks, using a *hierarchical path selection scheme* (cf. Section 2.1) that guarantees a competitive ratio of $O(\log^3 n)$. Second, there is a path selection scheme with an improved competitive ratio of $O(\log n)$ [33]. The latter scheme can be roughly viewed as a convex combination of spanning trees.¹ A path between a vertex s and a vertex t is chosen by sampling a random spanning tree and then choosing the path between s and t in this tree.

In this paper, we will build upon the original result [32] which we call the *hierarchical path selection scheme*. The challenge with implementing the path selection mechanism in [33] space-efficiently is that the number of spanning trees is quite large (polynomial in n). It seems difficult to avoid that a vertex in the graph has to store some information for every tree, which yields routing tables of polynomial size. The approach in [32] is based on a single tree which hence avoids the problem of [33].

The hierarchical path selection scheme is based on a hierarchical decomposition of the graph $G = (V, E)$. The vertex set V is recursively partitioned into smaller and smaller pieces until all pieces contain just single vertices of G . We will refer to the pieces/subsets arising during this partitioning process as *clusters*.

To such a recursive partitioning corresponds a decomposition tree $T = (V_T, E_T)$. A vertex x in this tree corresponds to cluster $V_x \subseteq V$ and there is an edge between a parent node p and a child node c if the cluster V_c arises from partitioning V_p . The root r of T corresponds to the subset $V_r = V$ and the leaf vertices correspond to singleton sets $\{v\}, v \in V$.

In order to simplify the notation and description we assume that all leaf vertices in T have the same distance to the root (this could e.g., be achieved by introducing dummy partitioning steps in which a set is partitioned into itself). We use h to denote the height of the tree. Let for a vertex $v \in V$, $a_\ell(v)$ denote the ancestor of $\{v\}$ on level ℓ of the tree, where the level of a vertex is its distance from the root. Here we use $\{v\}$ as a shorthand for “the leaf node that corresponds to cluster $\{v\}$ ”. The ℓ -weight of v is the weight of all edges incident to v that leave the cluster $V_{a_\ell(v)}$. Formally $w_\ell(v) := \sum_{e=\{v,x\}:x \notin V_{a_\ell(v)}} \text{cap}(e)$. We extend this definition to subsets of V by setting $w_\ell(U) := \sum_{u \in U} w_\ell(u)$ for every subset $U \subseteq V$.

We also introduce for every cluster S in the decomposition tree a weight function $w_S : S \mapsto \mathbb{R}_0^+$ and a weight function $\text{out}_S : S \mapsto \mathbb{R}_0^+$. For a level ℓ -cluster S we define $w_S := w_{\ell+1}|_S$ and $\text{out}_S := w_\ell|_S$, i.e., we define it as the restriction of $w_{\ell+1}$ and w_ℓ , respectively, to the vertex set of cluster S . Note that out_S counts edges that connect vertices of S to vertices outside of S while w_S also counts edges that connect different sub-clusters of S . We refer to w_S as the *cluster-weight* of S and to out_S as the *border-weight* of S .

¹ This is not entirely correct as the trees are not proper spanning trees but the difference is not important for the above discussion.

Using this weight definition, we define a *concurrent multicommodity flow problem* (CMCF-problem) for every cluster S in the decomposition tree. For every (ordered) pair (u, v) there is a demand of $w_S(u)w_S(v)/w_S(S)$. Informally speaking, this means that every vertex injects a total flow that is equal to its w_S -weight and distributes this flow to the other vertices in S , proportionally to the w_S -weight of these vertices. We will use the decomposition tree T in [32] with the following properties:

- the height of T is $O(\log n)$, and
- for every cluster S in the decomposition tree, the CMCF-problem for S can be solved with congestion at most $C = O(\log^2 n)$ inside S .

Now suppose that we are given a decomposition tree with these properties. The path selection in [32] is then performed as follows. Suppose that we want to choose a path between vertices s and t in G . Let x_s and x_t denote the leaf vertices in T that correspond to singleton clusters $\{s\}$ and $\{t\}$, respectively. Let $x_s = x_1, x_2, \dots, x_k = x_t$ denote the vertices in the tree on the path from x_s to x_t . We first choose a random vertex v_i from each cluster V_{x_i} according to the cluster-weight, i.e., the probability that v is chosen is $w_{V_{x_i}}(v)/w_{V_{x_i}}(V_{x_i})$. Note that $v_1 = s$ and $v_k = t$ as the corresponding clusters just contain a single vertex. It remains to select a path that connects the chosen vertices.

Suppose we want to connect two consecutive vertices v_p and v_c , where V_{x_p} is the parent cluster of V_{x_c} . We choose an intermediate vertex α inside V_{x_c} according to the border-weight of V_{x_c} , i.e., the probability that v is chosen is $\text{out}_{V_{x_c}}(v)/\text{out}_{V_{x_c}}(V_{x_c})$. We then consider the solution to the CMCF-flow problems for V_{x_c} and V_{x_p} . The first solution contains a flow $f(c, \alpha)$ between v_{x_c} and α , and the second contains a flow $f(p, \alpha)$ between v_{x_p} and α . We sample a random path from each flow. Concatenating these two paths, gives a flow between v_c and v_p . For the following analysis we call the sub-path between x_c and α the *lower sub-path* and the path between α and x_p the *upper sub-path*.

Concatenating all vertices v_i in the above manner gives a path between x_s and x_t . In the following we analyze the expected load generated on an edge due to this path selection scheme under the condition that an optimal algorithm can route the demand with congestion C_{opt} . For completeness and as we will need to modify this proof later, we repeat the following observations from [32].

► **Lemma 2.** *The expected load on an edge is at most $O(h \cdot C \cdot C_{\text{opt}})$.*

Proof. Fix an edge e for which both end-points are contained in some cluster S . Let S_1, \dots, S_r denote the child-clusters of S . We first analyze the total demand that we have to route between a pair of vertices $(a, b) \in S \times S$ due to an upper sub-path where a is chosen as the intermediate vertex α and b is chosen as a random vertex from the parent cluster S . Assume $a \in S_i$ for some child cluster S_i . Then the probability that we choose a as α is $\Pr[a \text{ is chosen}] = \text{out}_{S_i}(a)/\text{out}_{S_i}(S_i)$. The probability that we choose b as the random end-point in S is $\Pr[b \text{ is chosen}] = w_S(b)/w_S(S)$. Note that any message for which we route between the child cluster S_i and the parent cluster S has to leave or enter the cluster S_i . Therefore the total demand for these messages can be at most $C_{\text{opt}} \cdot \text{out}_{S_i}(S_i)$, as otherwise an optimum congestion of C_{opt} would not be possible. Hence, the expected demand for pair a and b is only

$$\begin{aligned} \text{out}_{S_i}(S_i)C_{\text{opt}} \cdot \Pr[a \text{ is chosen}] \cdot \Pr[b \text{ is chosen}] &= \text{out}_{S_i}(S_i)C_{\text{opt}} \cdot \frac{\text{out}_{S_i}(a)}{\text{out}_{S_i}(S_i)} \cdot \frac{w_S(b)}{w_S(S)} \\ &= \frac{w_S(a) \cdot w_S(b)}{w_S(S)} \cdot C_{\text{opt}}, \end{aligned} \quad (1)$$

where we used the fact that $\text{out}_{S_i}(a) = w_S(a)$, which holds since S_i is a direct child-cluster of S .

Now we analyze the demand that is induced for a pair $(a, b) \in S \times S$ due to the lower part of a message between S and its parent cluster. We assume that a is chosen as the intermediate vertex α and b is chosen as a random node in the child-cluster S . The probability that a is chosen as intermediate vertex is $\Pr[a \text{ is chosen}] = \text{out}_S(a)/\text{out}_S(S)$ and the probability that b is chosen is $\Pr[b \text{ is chosen}] = w_S(b)/w_S(S)$. Every such message has either to leave or enter cluster S . Hence, the total demand for these messages induced on pair (a, b) is at most

$$\begin{aligned} \text{out}_S(S)C_{\text{opt}} \cdot \Pr[a \text{ is chosen}] \cdot \Pr[b \text{ is chosen}] &= \text{out}_S(S)C_{\text{opt}} \cdot \frac{\text{out}_S(a)}{\text{out}_S(S)} \cdot \frac{w_S(b)}{w_S(S)} \\ &\leq \frac{w_S(a) \cdot w_S(b)}{w_S(S)} \cdot C_{\text{opt}} \quad , \end{aligned} \quad (2)$$

where we used the fact that $\text{out}_S(a) \leq w_S(a)$.

Combining Equation 1 and Equation 2 gives that the messages involving cluster S induce a demand of only $2w_S(a) \cdot w_S(b)/w_S(S) \cdot C_{\text{opt}}$ between vertices a and b from S . Since we route this demand according to the multicommodity flow solution of the CMCF-problem for cluster S , the resulting load is at most $2C \cdot C_{\text{opt}}$ on any edge *inside* cluster S , while edges not in S have a load of zero. Summing the load induced by messages for all clusters and exploiting the fact that an edge is at most contained in h different clusters, gives a maximum load of $2hC \cdot C_{\text{opt}}$, i.e., a competitive ratio of $2hC$. ◀

Harrelson, Hildrum and Rao [21] present a decomposition tree in which the congestion for the CMCF-problem of clusters is not uniformly bounded by C but it is guaranteed that along a root-to-leaf path the congestion values of the respective flow problems sum up to at most $O(\log^2 n \log \log n)$. Then the expected load in Lemma 2 becomes $O(\log^2 n \log \log n \cdot C_{\text{opt}})$. In addition the construction of this decomposition tree is polynomial time.

In the following description we base our oblivious routing scheme on the results in [32] as it slightly simplifies the write-up. For the theorems we also present the improved version obtained by plugging in the decomposition tree from Harrelson et al.

2.2 Implementation A: Decomposition Trees with Small Degree

We now present a space efficient implementation of the above path selection scheme. In the following, we will assume that the maximum degree of the decomposition tree T is small.

The basic building block for our implementation is a method that given a random starting point $v \in S$ chosen according to the cluster-weight of S (i.e., the probability of choosing v is $w_S(v)/w_S(S)$), routes to a random node $v_i \in S_i$ chosen according to the border weight of S_i . Here S_i is either a child-cluster of S (in case we want to communicate downwards in the tree) or $S_i = S$ (in case we want to communicate upwards). In the following we use $S_i, i \in \{1, \dots, r\}$ to denote the child-clusters of S and $S_0 = S$ to denote S itself. Let $G[S]$ denote the sub-graph induced by vertices in S .

For every $i \in \{0, \dots, r\}$ we compute a single commodity flow f_i in $G[S]$ as follows. We add a super-source s and connect it to every vertex $v \in S$ with an edge of capacity $w_S(v) \cdot \text{out}_{S_i}(S_i)$ and a super-target t to which every vertex in $v \in S$ connects with capacity $\text{out}_{S_i}(v) \cdot w_S(S)$. Note that all source edges together have the same capacity as the target edges.

We now compute a flow f_i between s_i and t_i that saturates all edges from s_i and to t_i . We can find such flows f_i (for all i) such that the *combined* congestion for these flows (on edges in S ; edges from s_i 's and to t_i 's have congestion 1) is only $w_S(S) \cdot C$. To see this observe that in the CMCF-solution for cluster S the commodity (v_i, v) with $v_i \in S_i$ and $v \in S$ ships a flow of $\text{out}_{S_i}(v_i)w_S(v)/w_S(S)$ between v_i and v . By “merging” the flows of all

commodities $(v_i, v) \in S_i \times S$ into a single commodity we obtain the desired flow (up to a $w_S(S)$ -factor). Merging the commodities does not increase the congestion and, hence, the congestion is only $w_S(S) \cdot C$.

The flows f_i that we constructed so far may have fractional values that are difficult to store exactly. Therefore we slightly change the flows so that we can store them efficiently. For this we first scale every flow and the capacity of every edge up by a factor of r . Let f'_i and $\text{cap}'(e)$ denote the scaled flows and capacities. Then every edge e makes a *capacity reservation* for every flow f_i . Suppose the (scaled) flow sends $f'_i(e)$ along edge e ; then the edge e reserves a capacity of $\lceil f'_i(e) \rceil$ for the i -th flow. Note that the total capacity reservation is at most $\sum_i \lceil f'_i(e) \rceil \leq \sum_i f'_i(e) + r \leq w_S(S) \cdot C \text{cap}'(e) + r \leq 2w_S(S) \cdot C \text{cap}'(e)$, because the scaled capacity of an edge is at least r .

Now we resolve every flow problem separately with the restriction that the flow should stay within its capacity reservation. This is clearly possible and since the capacity reservations and the demands are all integral we now have an *integral* flow f'_i . Undoing the scaling gives us flows f_i that can (concurrently) be routed with congestion at most $2w_S(S) \cdot C$, and where flow values are a multiple of $1/r$.

We now store the flows f_i in a distributed manner at the vertices of S , as follows. Fix $v \in S$. For every edge we store how much flow enters or leaves v . In order to route from the cluster-distribution of S to the border-distribution for S_i , $i \in \{0, \dots, r\}$, we choose random outgoing links (where a link is taken with probability proportional to the outgoing flow) until the chosen link is the super-target t . When we want to route from the border-distribution of S_i to the cluster-distribution of S , we take random incoming links (where a link is chosen with probability proportional to the incoming flow), until the chosen link corresponds to the super-source s . The proof of the following claim is analogous to Lemma 2.

▷ **Claim 3.** The expected load of an edge due to the path selection scheme is only $O(h \cdot C \cdot C_{\text{opt}})$.

Proof. Suppose that the optimum congestion is C_{opt} . The total traffic that the scheme has to route between the cluster-distribution of S and the border-distribution of S_i is only $\text{out}_{S_i}(S_i) \cdot C_{\text{opt}}$. We route this traffic according to flow f_i of value $\text{out}_{S_i}(S_i)w_S(S)$. Hence, the maximum load of an edge in $G[S]$ (according to original capacity) is $C \cdot C_{\text{opt}}$.

Since an edge is contained in h different clusters the claim follows. ◁

▷ **Claim 4.** The path selection scheme can be implemented with routing tables of size $O(\deg(v) \deg(T)(\log m + \log W))$, labels of length $O(h \log(\deg(T)))$, and header length $O(h \log(\deg(T)))$.

Proof. Suppose that the capacities of the graph are integers in the range $\{1, \dots, W\}$. A flow value $f_i(e)$ along an edge is at most $w_S(S) \cdot W \cdot C$ (note that we assume that C is integral). Edges from s and to t have a capacity of $w_S(v) \text{out}_{S_i}(S_i)$ and $w_S(S) \text{out}_{S_i}(v)$, respectively. Using the fact that $w_S(S)$ and $\text{out}_{S_i}(S_i)$ are at most mW , and $d, C \leq m$ we get that a number describing the flow value along an edge can be encoded with

$$\log_2(m^2 W^2 r) = O(\log(m) + \log(W) + \log(r))$$

many bits (since flow values are a multiple of $1/r$). Hence, a node v has to store only $O(\deg(v) \deg(T)(\log m + \log W))$ many bits, where we used that $r \leq m$.

For the routing scheme we relabel the vertices. We number the children of a vertex in the tree and encode a leaf vertex by its path from the root. This generates labels of $O(h \log(\deg(T)))$ bits. The routing algorithm now only needs to have the label of the source vertex and the target vertex and a marker that marks where in the tree the routing currently is. ◁

In summary, we obtain the following theorem.

► **Theorem 5** (Decomposition Trees of Small Degree). *For decomposition trees of degree $\deg(T)$ one can construct an oblivious routing strategy that requires routing tables of size $O(\deg(v) \deg(T)(\log(m) + \log(W)))$, labels of length $O(h \log(\deg(T)))$, and header sizes of $O(h \log(\deg(T)))$. Depending on the decomposition tree used, we obtain two different competitive ratios:*

- *Using the decomposition tree from [32] the scheme guarantees a competitive ratio of $O(hC) = O(\log^3(n))$ w.r.t. congestion.*
- *Using the decomposition tree from [21] the scheme can be constructed in polynomial time and guarantees a competitive ratio of $O(\log^2(n) \log \log(n))$.*

2.3 Implementation B: Uniform Capacities

In this section we present a different implementation of the hierarchical routing scheme, for scenarios where the decomposition trees can be of arbitrary degree but where network capacities are uniform. Again the basic building block is to route from a node chosen according to the cluster-distribution of some cluster S to the border distribution of S_i where either $S_i = S$ or S_i is a child-cluster of S .

Assume that every edge in the graph G has capacity 1. We round the outgoing capacity $\text{out}_{S_i}(S_i)$ of a child-cluster S_i , $i \geq 0$ to the next larger power of 2 and denote the rounded value with $\|S_i\|$. We also re-order the children w.r.t. this value, i.e., S_1 is the child-cluster with smallest $\|S_i\|$ -value. Since there are at most m possible values for $\text{out}_{S_i}(S_i)$, there are only $\log m$ possible values for $\|S_i\|$. There are only $\binom{r + \log m}{\log m}$ possibilities to choose the $\|S_i\|$ -values of the r children of cluster S . Hence, we can store these with $O(\log(m) \cdot \log(r))$ many bits. In addition we store the value of $\|S_0\|$, which requires $O(\log \log m)$ bits, and the value of $w_S(S)$ which requires $O(\log m)$ bits.

In order to design the routing scheme for an individual cluster, we embed a hypercube of dimension $d := \lceil \log_2(\sum_{i \geq 0} \|S_i\|) \rceil$. We first order the hypercube nodes in an arbitrary way and then reserve a range of $\|S_i\|$ consecutive hypercube nodes for every $i \geq 0$ (the i -th range). Note that we store the (rounded) size of all children and that it is straightforward to compute the ranges assigned for any i from this information.

Then we map the hypercube nodes to nodes of S . First we map hypercube nodes in the i -th range to nodes with $\text{out}_{S_i}(v) > 0$ such that each node receives at least $\text{out}_{S_i}(v)$ and at most $2 \text{out}_{S_i}(v)$ hypercube nodes. Hypercube nodes that remain unmapped after this step (i.e., nodes that do not fall within any range) are mapped arbitrarily subject to the constraint that a cluster node v does not receive more than $8w_S(v)$ hypercube nodes. This can easily be done as the number of hypercube nodes (2^d) is at most $2 \sum_i \|S_i\| \leq 4 \sum_i \sum_{v \in S_i} \text{out}_{S_i}(v) = 4(w_S(S) + \text{out}_S(S)) \leq 8w_S(S)$.

► **Observation 6.** *There are at most $8w_S(v)$ hypercube nodes mapped to any graph node.*

For the embedding we set up a concurrent multicommodity flow problem as follows. For every edge $\{x, y\}$ of the hypercube that is mapped to endpoints $\{v_x, v_y\}$, we introduce a demand of 1 between v_x and v_y in both directions. Then every node sends and receives a total traffic of at most $8d \cdot w_S(v)$. By adding fake traffic we can turn this instance into a balanced multicommodity flow instance in which every vertex sends and receives a traffic of exactly $8d \cdot w_S(v)$.

We can solve this multicommodity flow instance with congestion at most $16dC$ inside the cluster S by using Valiant's trick [38, 25] of sending to random intermediate destinations and using the fact that each flow can send a traffic of $w_S(v)$ to random destinations with congestion C .

2.3.1 Using the Hypercube

How do we exploit the embedded hypercube? If during the routing scheme we are required to send a message from a cluster node v_p to a cluster node $v_c \in S_i$ we proceed as follows. Instead of choosing an intermediate node α according to probability distribution $\text{out}_{S_i}(v)/\text{out}_{S_i}(S_i)$ we choose a random hypercube node from the i -th range. Then we route a message inside the hypercube to this node. For this we let the message start at a random hypercube node from the nodes that are mapped to v_p .

Note that this means that the probability that the message is sent to node α lies between $\text{out}_{S_i}(\alpha)/\|S_i\|$ and $2\text{out}_{S_i}(\alpha)/\|S_i\|$ as the hypercube nodes in the i -th range are not mapped completely uniformly.

For the second part of the message we proceed analogously in the hypercube of S_i . We let the message start at a random hypercube node mapped to α and choose a random hypercube node as its target.

Again due to the non-uniform mapping, the target distribution on S_i (i.e., $w_{S_i}(v)/w_{S_i}(S_i)$) is not reached exactly, but deviations by a constant factor might occur. This only influences the congestion of a single step by a constant factor, but it could be problematic if we used this approach along a path in the tree: in each step the distribution would change by a constant factor.

Therefore, we add an additional step that fixes the distribution over S_i . We embed an additional hypercube H_S for every cluster S with dimension $\lceil \log_2(w_S(S)) \rceil$. The mapping is done such that each cluster-vertex $v \in S$ receives exactly $w_S(v)$ hypercube nodes among the first $w_S(S)$ nodes from H_S (the remaining nodes are distributed uniformly). Since every node in the cluster S stores the value of $w_S(S)$, we can route from a node $v \in S$ to a random node chosen according to distribution $w_S(v)/w_S(S)$, by just selecting a random hypercube node from the first $w_S(S)$ nodes.

2.3.2 Analysis

We showed that the congestion for sub-messages that involve cluster S is small. There are two types of such messages:

1. messages that start at an intermediate node (distributed according to the border weight of S_i for some $i \geq 0$) and are sent to a random node $v \in S$ distributed according to the cluster-weight of S ; and
2. messages that start at a random node $v \in S$ and are sent to some intermediate node.

It was shown that the total traffic that is sent between a pair v_i and v , where v is distributed according to the cluster weight of S and v_i is distributed according to the border weight of S_i , is only $\text{out}_{S_i}(v_i)w_S(v)/w_S(S) \cdot C_{\text{opt}}$.

In our new scheme this changes slightly. For messages of the second type the source is distributed as before but the target may have a slightly different distribution (as we choose a random hypercube node in the i -th range). For messages of the first type already the source may have a slightly different distribution (as we choose a random hypercube node from some range in the hypercube for a child- or parent-cluster). Also the target distribution is slightly skewed as we choose a random hypercube node as the target.

But since the distributions are only changed by a constant factor this difference does not really influence our analysis. We still have the property that the traffic between v_i and v_S is $\Theta(\text{out}_{S_i}(v_i)w_S(v)/w_S(S) \cdot C_{\text{opt}})$.

The second difference is that the traffic is not sent according to the CMCF-problem for cluster S but it is instead sent along the hypercube. Note that due to the embedding of the hypercube, a cluster node $v \in S_i$ has $\Theta(\text{out}_{S_i}(v)) = \Theta(w_S(v))$ hypercube nodes in the i -th

range mapped to it (i.e., hypercube nodes are balanced perfectly up to constant factors). Hence the demand between v_i and v will be split among $\Theta(\text{out}_{S_i}(v_i)w_S(v))$ pairs in the cube. Therefore the demand for every pair in the cube is only $\Theta(C_{\text{opt}}/w_S(S)) = \Theta(C_{\text{opt}}/2^d)$. This means that at most a traffic of $O(C_{\text{opt}})$ starts and ends at every vertex and routing this traffic using Valiant's trick gives a congestion of $O(dC_{\text{opt}})$ in the hypercube. Since we embedded the hypercube with congestion $O(dC)$, the congestion of a graph edge will be $O(d^2C \cdot C_{\text{opt}})$ (as each hypercube node has degree d), which gives rise to the following lemma.

► **Lemma 7.** *Implementation B guarantees a maximum expected load of $O(hd^2C \cdot C_{\text{opt}})$.*

Proof. The lemma follows by applying the previous argument for each level of the tree.

It remains to bound the edge-load induced by the re-randomization steps. The total traffic that is sent to a cluster S in the tree is at most $(\sum_i \text{out}(S_i)) \cdot C_{\text{opt}} = \Theta(w_S(S) \cdot C_{\text{opt}})$. For each such message we require a re-randomization, because in our current scheme, it is only distributed approximately according to the cluster-weight of S .

However by design each vertex receives exactly a $w_S(v)/w_S(S)$ -fraction of the re-randomization messages, and a $\Theta(w_S(v)/w_S(S))$ -fraction of messages start at v , since the messages are approximately distributed according to cluster-weight. Sending these messages along the hypercube introduces congestion $\Theta(d \cdot C_{\text{opt}})$ in the cube and $\Theta(d^2C \cdot C_{\text{opt}})$ due to the embedding. ◀

► **Lemma 8.** *Implementation B requires space $O(hC \log(m) \log \log(m) \deg(v))$ bits at every vertex and a label and header length of $O(h \log(\deg(T)))$.*

Proof. We will use the following helper lemma:

► **Lemma 9.** *Let X_1, \dots, X_n denote a set of negatively correlated random variables taking values in the range $[0, 1]$. Let X denote their sum and let $\mu \leq E[X]$ denote a lower bound on the expectation of X . Then for any $\delta \geq 1$*

$$\Pr[X \geq (1 + \delta)\mu] \leq e^{-\delta\mu/3} .$$

A vertex $v \in S$ has to store the approximate size $\|S_i\|$ of the child-clusters of S . Summing this over all levels gives $O(h \log(m) \cdot \log(r))$ bits. In addition one has to encode the embedding of the hypercubes. The congestion of the solution to the concurrent multicommodity flow problem for embedding a hypercube is $O(dC)$. This fractional solution will encode a flow for every hypercube edge. Using a standard randomized rounding approach, we can route the flows to paths with a congestion of $O(dC + \log(m)) = O(dC)$. This is done as follows. For every pair $\{x, y\}$ we take the unit flow and first decompose this unit flow into flow-paths. Then we choose for every pair one of the flow-paths at random (proportional to its weight). Let $X_i(e)$ denote the random variable that describes whether the flow path for the i -th pair includes edge e . By design the above process guarantees $E[X_i(e)] = f_i(e)$, where $f_i(e)$ is the flow for pair i that goes through edge e . The total load on edge e is $\sum_i X_i(e)$. This is a sum of negatively correlated random variables with expectation $\mu = O(dC)$. Using Lemma 9 with $\delta = 3 \ln(m)/\mu$ gives that with constant probability, no edge exceeds load $O(dC + \log(m))$.

Therefore only $O(\deg(v)dC)$ paths traverse a vertex v (recall that $C \geq \log(m)$). For every path, we need to store the outgoing edge and the id of the paths on this edge. This requires $(\log_2(\deg(v)) + \log_2(dC))$ bits for every path and $O(d \deg(v)C \log(d \deg(v)C))$ bits in total. Multiplying with the height and using $d = \Theta(\log(m))$ gives $O(hC \log(m)(\log(\deg(v)) + \log \log(m) \deg(v))) = O(hC \log^2(m) \cdot \deg(v))$ bits.

75:10 Compact Oblivious Routing

The header- and label-length is analogous to Implementation A. We just use the root-to-leaf path in the tree as a label and a header consists of the source-label, the target-label, and a marker. ◀

In summary we derived the following result:

► **Theorem 10** (Compact Oblivious Routing for Uniform Capacities). *For arbitrary networks of uniform capacities, there exists an oblivious routing strategy which requires label and header length of $O(h \log(\deg(T)))$, and which comes in two flavors, depending on the decomposition tree used:*

- *Using the decomposition tree from [32] the scheme requires $O(hC \log^2(m) \deg(v)) = O(\log^5(n) \deg(v))$ bits at every vertex and guarantees a competitive ratio of $O(hC \log^2(n)) = O(\log^5(n))$ w.r.t. congestion.*
- *Using the decomposition tree from [21] the scheme requires $O(\log^2 m \log^2(n) \log \log n) \deg(v) = O(\log^4 n \log \log n) \deg(v)$ bits at every vertex and guarantees a competitive ratio of $O(\log^4(n) \log \log(n))$ w.r.t. congestion. The oblivious routing scheme can be constructed in polynomial time.*

3 Related Work

The drawbacks of *adaptive routing* have been discussed intensively in the literature, see e.g., [34] for a survey. In particular, adaptive routing schemes need global information about the routing problem in order to calculate the best paths, and even if it were possible to collect such information sufficiently fast, it can still take much time to compute a (near-)optimal solution to that problem (large linear programs may have to be solved).

One of the first and well-known results on *oblivious routing* is due to Borodin and Hopcroft [8] who showed that competitive oblivious routing algorithms require randomization, as deterministic algorithms come with high lower bounds: given an unweighted network with n nodes and maximum degree Δ , there exists a (permutation) routing instance such that the congestion induced by a given deterministic oblivious routing scheme is at least $\Omega(\sqrt{n}/\Delta^{3/2})$. This result was improved by Kakkamanis et al. [22] to a lower bound of $\Omega(\sqrt{n}/\Delta)$.

For randomized algorithms Valiant and Brebner [38] showed how to obtain a polylogarithmic competitive ratio for the hypercube by routing to random intermediate destinations. Räcke [32] presented the first oblivious routing scheme with a polylogarithmic competitive ratio of $O(\log^3 n)$ in general networks. The paper by Räcke was also the first to propose designing oblivious routing schemes based on cut-based *hierarchical decompositions*. However, Räcke's result is non-constructive in the sense that only an exponential time algorithm was given to construct the hierarchy. This approach has subsequently been used to obtain approximate solutions for a variety of cut-related problems that seem very hard on general graphs but that are efficiently solvable on trees, see e.g. [2, 3, 5, 10, 14, 23, 26, 33]. Polynomial-time algorithms for constructing the hierarchical decomposition were given by Bienkowski et al. [7] and Harrelson et al. [21]. However, none of these results provide an (asymptotically) optimal competitive ratio.

Azar et al. [4] gave a polynomial time algorithm that for a given graph computes the optimal oblivious routing via a linear programming approach, i.e., without using a hierarchical decomposition.

An optimal competitive ratio of $O(\log n)$ (which matches a known lower bound from grids [6, 30]) was first presented by Räcke [33]. Instead of considering a single tree to approximate the cut-structure of a graph G , [33] proposes to use a convex combination

of decomposition trees. The paper relies on multiplicative weight updates and the proof technique is similar to the technique used by Charikar et al. [9] for finding a probabilistic embedding of a metric into a small number of dominating tree metrics.

More recently, inspired by the ideas on cut matching games introduced by Khandekar, Rao, and Vazirani [24], Räcke et al. [35] presented a *fast* construction algorithm for hierarchical tree decompositions, i.e., for a single tree: given an undirected graph $G = (V, E, c)$ with edge capacities, a single tree $T = (V_T, E_T, c_T)$ can be computed whose leaf nodes correspond to nodes in G and which approximates the cut-structure of G up to a factor of $O(\log^4 n)$ (i.e., the faster runtime comes at the price of a worse approximation guarantee). In particular, the authors present almost linear-time cut-based hierarchical decompositions, by establishing a connection between approximation of max flow and oblivious routing. This overcomes the major drawback of earlier algorithms such as [21] and even [29] which required high running times for constructing the decomposition tree (or the distribution over decomposition trees). The bound has been improved further by Peng in [31].

Previous results on *compact routing* focus on routing strategies that aim to minimize the *path length* instead of the congestion (see e.g. [13, 27, 39]). There are two variants: *labeled* (the designer is free to name the nodes according to the topology and the edge weights of the graph) and *name-independent* (name determined by an adversary). The research community has derived many interesting results on compact shortest path routing on special graphs, e.g., characterizing hypercubes, trees, scale-free networks, and planar graphs [17, 18, 19, 28, 37]. A well-known recent result on compact routing in the name-independent model on *general graphs* is by Abraham et al. [1].

However, it is also known that it is impossible to implement shortest path routing with routing tables whose size in all network topologies grows slower than linear with the increase of the network size [16, 20]. As a resort, compact routing research studies algorithms to decrease routing table sizes at the price of letting packets to be routed along suboptimal paths. In this context, suboptimal means that the forwarding paths are allowed to be longer than the shortest ones, but the length increase is bounded by a constant stretch factor. A particularly interesting result is by Thorup et al. [37] who presented compact routing schemes for general weighted undirected networks, ensuring small routing tables, small headers and low stretch. The approach relies on an interesting shortest path routing scheme for trees of arbitrary degree and diameter that assigns each vertex of an n -node tree a label of logarithmic size. Given the label of a source node and the label of a destination it is possible to compute, in constant time, the port number of the edge from the source that heads in the direction of the destination. An interesting recent work by Retvari et al. [36] generalizes compact routing to arbitrary routing policies that favor a broader set of path attributes beyond path length. Using routing algebras, the authors identify the algebraic requirements for a routing policy to be realizable with sublinear size routing tables.

There also exist interesting works focusing on *scalable* oblivious traffic engineering, e.g., for confluent (per-destination) routing schemes [11, 12]. However, we are not aware of any results on compact oblivious routing which provides polylogarithmic approximation ratios along both dimensions, table size and congestion.

4 Conclusion

Given the fast growth of communication networks (e.g., due the advent of novel paradigms such as Internet-of-Things), the high costs of network equipment (e.g., fast memory is expensive and power hungry), as well as the increasing miniaturization of communication-

enabled devices, we in this paper initiated the study of oblivious routing schemes which only require small routing tables. In particular, we presented the first *compact* oblivious routing scheme, requiring polylogarithmic tables only (as well as polylogarithmic packet headers and vertex labels).

We believe that our work opens an interesting avenue for future research. In particular, while our algorithms provide poly-logarithmic routing tables and competitive ratios, it may be possible to further improve these results by logarithmic factors. Furthermore, it would be interesting to generalize our results to non-uniform network capacities, as well as to explore whether our results can be improved for special network topologies arising in practice.

References

- 1 Ittai Abraham, Cyril Gavoille, and Dahlia Malkhi. On Space-stretch Trade-offs: Upper Bounds. In *Proc. 18th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 217–224, 2006.
- 2 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Seffi Naor. A general approach to online network optimization problems. *ACM Transactions on Algorithms (TALG)*, 2(4):640–660, 2006.
- 3 Konstantin Andreev, Charles Garrod, Daniel Golovin, Bruce Maggs, and Adam Meyerson. Simultaneous source location. *ACM Transactions on Algorithms (TALG)*, 6(1):16, 2009.
- 4 Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Räcke. Optimal oblivious routing in polynomial time. *Journal of Computer and System Sciences*, 69(3):383–394, 2004.
- 5 Nikhil Bansal, Uriel Feige, Robert Krauthgamer, Konstantin Makarychev, Viswanath Nagarajan, Joseph Naor, and Roy Schwartz. Min-max graph partitioning and small set expansion. In *Proc. IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 17–26. IEEE, 2011.
- 6 Yair Bartal and Stefano Leonardi. On-line routing in all-optical networks. In *Proc. International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 516–526. Springer, 1997.
- 7 Marcin Bienkowski, Mirosław Korzeniowski, and Harald Räcke. A practical algorithm for constructing oblivious routing schemes. In *Proc. 15th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 24–33. ACM, 2003.
- 8 Allan Borodin and John E. Hopcroft. Routing, merging, and sorting on parallel models of computation. *Journal of computer and system sciences*, 30(1):130–145, 1985.
- 9 Moses Charikar, Chandra Chekuri, Ashish Goel, Sudipto Guha, and Serge Plotkin. Approximating a finite metric by a small number of tree metrics. In *Proc. 39th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 379–388. IEEE, 1998.
- 10 Chandra Chekuri, Sanjeev Khanna, and F Bruce Shepherd. The all-or-nothing multicommodity flow problem. In *Proc. 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 156–165. ACM, 2004.
- 11 Marco Chiesa, Gábor Rétvári, and Michael Schapira. Lying Your Way to Better Traffic Engineering. In *Proc. ACM 12th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 391–398, 2016.
- 12 Marco Chiesa, Gábor Rétvári, and Michael Schapira. Oblivious Routing in IP Networks. *IEEE/ACM Transactions on Networking (TON)*, 26(3):1292–1305, 2018.
- 13 Lenore J Cowen. Compact routing with minimum stretch. *Journal of Algorithms*, 38(1):170–183, 2001.
- 14 Roe Engelberg, Jochen Könemann, Stefano Leonardi, and Joseph Seffi Naor. Cut problems in graphs with a budget constraint. In *Proc. Latin American Symposium on Theoretical Informatics (LATIN)*, pages 435–446. Springer, 2006.
- 15 Klaus-Tycho Foerster, Stefan Schmid, and Stefano Vissicchio. Survey of Consistent Software-Defined Network Updates. In *IEEE Communications Surveys and Tutorials (COMST)*, 2018.

- 16 Pierre Fraigniaud and Cyril Gavoille. Memory requirement for universal routing schemes. In *Proc. 14th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 223–230. ACM, 1995.
- 17 Pierre Fraigniaud and Cyril Gavoille. Routing in trees. In *Proc. International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 757–772. Springer, 2001.
- 18 Greg N Frederickson and Ravi Janardan. Designing networks with compact routing tables. *Algorithmica*, 3(1-4):171–190, 1988.
- 19 Cyril Gavoille. Routing in distributed networks: Overview and open problems. *ACM SIGACT News*, 32(1):36–52, 2001.
- 20 Cyril Gavoille and Stéphane Pérennès. Memory requirement for routing in distributed networks. In *Proc. 15th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 125–133. ACM, 1996.
- 21 Chris Harrelson, Kirsten Hildrum, and Satish Rao. A polynomial-time tree decomposition to minimize congestion. In *Proc. 15th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 34–43, 2003.
- 22 Christos Kaklamanis, Danny Krizanc, and Thanasis Tsantilas. Tight bounds for oblivious routing in the hypercube. *Mathematical Systems Theory*, 24(1):223–232, 1991.
- 23 Rohit Khandekar, Guy Kortsarz, and Vahab Mirrokni. On the advantage of overlapping clusters for minimizing conductance. *Algorithmica*, 69(4):844–863, 2014.
- 24 Rohit Khandekar, Satish Rao, and Umesh Vazirani. Graph partitioning using single commodity flows. *Journal of the ACM (JACM)*, 56(4):19, 2009.
- 25 Petr Kolman and Christian Scheideler. Improved bounds for the unsplittable flow problem. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 184–193. Society for Industrial and Applied Mathematics, 2002.
- 26 Jochen Könemann, Ojas Parekh, and Danny Segev. A unified approach to approximating partial covering problems. In *Proc. European Symposium on Algorithms (ESA)*, pages 468–479. Springer, 2006.
- 27 Dmitri Krioukov, Kevin Fall, Arthur Brady, et al. On compact routing for the internet. *ACM SIGCOMM Computer Communication Review (CCR)*, 37(3):41–52, 2007.
- 28 Dmitri Krioukov, Kevin Fall, and Xiaowei Yang. Compact routing on Internet-like graphs. In *Proc. IEEE INFOCOM*. IEEE, 2004.
- 29 Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *Proc. 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 245–254. IEEE, 2010.
- 30 Bruce M Maggs, F Meyer auf der Heide, Berthold Vocking, and Matthias Westermann. Exploiting locality for data management in systems of limited bandwidth. In *Proc. 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 284–293. IEEE, 1997.
- 31 Richard Peng. Approximate undirected maximum flows in $o(m \text{ polylog}(n))$ time. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1862–1867. Society for Industrial and Applied Mathematics, 2016.
- 32 Harald Räcke. Minimizing Congestion in General Networks. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 43–52, 2002. doi: 10.1109/SFCS.2002.1181881.
- 33 Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proc. 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 255–264. ACM, 2008.
- 34 Harald Räcke. Survey on Oblivious Routing Strategies. In *Proc. 5th Conference on Computability in Europe: Mathematical Theory and Computational Practice (CiE)*, pages 419–429, 2009.
- 35 Harald Räcke, Chintan Shah, and Hanjo Täubig. Computing cut-based hierarchical decompositions in almost linear time. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 227–238. Society for Industrial and Applied Mathematics, 2014.

75:14 Compact Oblivious Routing

- 36 Gábor Rétvári, András Gulyás, Zalán Heszberger, Márton Csernai, and József J Bíró. Compact policy routing. *Distributed computing*, 26(5-6):309–320, 2013.
- 37 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proc. 19th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. ACM, 2001.
- 38 Leslie G. Valiant and Gordon J. Brebner. Universal Schemes for Parallel Communication. In *Proceedings of the 13th ACM Symposium on Theory of Computing (STOC)*, pages 263–277, 1981. doi:10.1145/800076.802479.
- 39 Jan van Leeuwen and Richard B Tan. Compact routing methods: A survey. In *Proc. Colloquium on Structural Information and Communication Complexity (SICC)*, pages 99–109, 1995.

Geometric Crossing-Minimization – A Scalable Randomized Approach

Marcel Radermacher

Department of Computer Science, Karlsruhe Institute of Technology, Germany
radermacher@kit.edu

Ignaz Rutter

Department of Computer Science and Mathematics, University of Passau, Germany
rutter@fm.uni-passau.de

Abstract

We consider the minimization of edge-crossings in geometric drawings of graphs $G = (V, E)$, i.e., in drawings where each edge is depicted as a line segment. The respective decision problem is \mathcal{NP} -hard [5]. Crossing-minimization, in general, is a popular theoretical research topic; see Vrt'o [26]. In contrast to theory and the topological setting, the geometric setting did not receive a lot of attention in practice. Prior work [21] is limited to the crossing-minimization in geometric graphs with less than 200 edges. The described heuristics base on the primitive operation of moving a single vertex v to its *crossing-minimal position*, i.e., the position in \mathbb{R}^2 that minimizes the number of crossings on edges incident to v .

In this paper, we introduce a technique to speed-up the computation by a factor of 20. This is necessary but not sufficient to cope with graphs with a few thousand edges. In order to handle larger graphs, we drop the condition that each vertex v has to be moved to its crossing-minimal position and compute a position that is only optimal with respect to a small random subset of the edges. In our theoretical contribution, we consider drawings that contain for each edge $uv \in E$ and each position $p \in \mathbb{R}^2$ for v $o(|E|)$ crossings. In this case, we prove that with a random subset of the edges of size $\Theta(k \log k)$ the *co-crossing* number of a degree- k vertex v , i.e., the number of edge pairs $uv \in E, e \in E$ that do *not* cross, can be approximated by an arbitrary but fixed factor δ with high probability. In our experimental evaluation, we show that the randomized approach reduces the number of crossings in graphs with up to 13 000 edges considerably. The evaluation suggests that depending on the degree-distribution different strategies result in the fewest number of crossings.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Mathematics of computing \rightarrow Graph algorithms

Keywords and phrases Geometric Crossing Minimization, Randomization, Approximation, VC-Dimension, Experiments

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.76

Related Version A full version of the paper is available at <https://arxiv.org/abs/1907.01243>.

Funding Work was supported by grants RU 1903/3-1 and WA 654/21-1 of the German Research Foundation (DFG).

1 Introduction

The minimization of crossings in geometric drawings of graphs is a fundamental graph drawing problem. In general the problem is \mathcal{NP} -hard [5, 13] and has been studied from numerous theoretical perspectives; see Vrt'o [26]. Until recently only the topological setting, where edges are drawn as topological curves, has been considered in practice [6, 8, 14]. In our previous paper [21] we describe geometric heuristics that compute straight-line drawings of graphs with significantly fewer crossings compared to common energy-based layouts. One of the heuristics is the *vertex-movement approach* that iteratively moves a single vertex v



© Marcel Radermacher and Ignaz Rutter;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 76; pp. 76:1–76:16



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to its *crossing-minimal position*, i.e., a position p^* so that crossings of edges incident to v are minimized. Unfortunately, the worst-case running time to compute this position is super-quadratic in the size of the graph as the following theorem states.

► **Theorem 1** (Radermacher et al. [21]). *The crossing-minimal position of a degree- k vertex v with respect to a straight-line drawing Γ of a graph $G = (V, E)$ can be computed in $O((kn + m)^2 \log(kn + m))$ time, where $n = |V|, m = |E|$.*

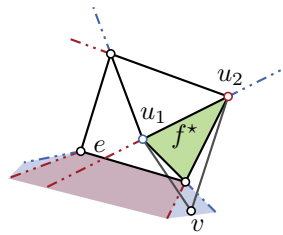
This is not only a theoretical upper bound on the running time but is also a limitation that has been observed in practice. The implementation we used previously requires considerable time to compute drawings with few crossings. For this reason we were only able to evaluate our approach on graphs with at most 200 edges. For example, on a class of graphs that have 64 vertices and 196 edges our implementation already required on average about 35 seconds to compute a drawing with few crossings.

Energy-based methods are common and well engineered tools to draw graphs [16]. For example, the aim of *Stress Majorization* (or simply STRESS) is to compute a drawing such that the Euclidean distance of each two vertices corresponds to their graph-theoretical distance [12]. The algorithm has been engineered to handle graphs with up to 10^6 vertices and $3 \cdot 10^6$ edges [19]. Kobourov [16] claimed that STRESS tends to minimize the number of crossings. In our previous experimental evaluation [21] we demonstrated that the statement is not true for a varied set of graph classes.

Fabila-Monroy and López [11] introduced a randomized algorithm to compute a drawing of K_n with a small number of crossings. Many best known upper bounds on the rectilinear crossing number of K_n , for $44 \leq k \leq 99$, are due to this approach [1]. The algorithm iteratively updates a set P of n points, by replacing a random point $p \in P$ by a random point q that is close to p , if q improves the number of crossings. Since the number of crossings of K_n is in $\Theta(n^4)$, the bottleneck of their approach is the running time for counting the number of crossings induced by P . A similar randomized approach has been used to maximize the smallest crossing angle in a straight-line drawing [3, 10]. The approach iteratively moves vertices to the best position within a random point set.

Contribution. The main contribution of this paper is to engineer the *vertex-movement approach* for the minimization of crossings in geometric drawings described in [21] to be applicable on graphs with a few thousands vertices and edges.

1. In Section 3 we introduce so-called *bloated duals of line arrangements*, a combinatorial technique to construct a dual representation of general line arrangements. In our application this results in an overall speed-up of about a factor of 20 in comparison to the recent implementation. This speed-up is necessary but not sufficient to handle graphs with a few thousands vertices and edges.
2. In Section 4 we demonstrate that taking a small random subset of the edges is sufficient to compute drawings with few crossings. Moreover, in Section 4.1 we prove that under certain conditions the randomized approach is an approximation of the co-crossing number of a vertex, with high probability.
3. Based on the insights of the evaluation in Section 4.2, we introduce a weighted sampling approach. A comparison to a restrictive approach of sampling points suggests that the degree-distribution of the graph is a good indicator to decide which approach results in fewer crossings.
4. Overall, our experimental evaluation shows that we are now able to handle graphs with 12 000 edges, which are 60 times more than the graphs that have been considered in the evaluation in [21].



■ **Figure 1** The black, blue and red segments show the arrangement $\mathcal{A}(\Gamma, v)$ of the black drawing Γ . The blue and red region show the complement of the visibility regions of u_1 and u_2 , respectively, and the edge e . The green region is crossing minimal.

2 Preliminaries

We repeat some notation from [21]. Let Γ be a straight-line drawing of a $G = (V, E)$. Denote by $N(v) \subseteq V$ the set of neighbors of v and by $E(v) \subseteq E$ the set of edges incident to v . For a vertex $v \in V$, denote by $\Gamma[v \mapsto p]$ the drawing that is obtained from Γ by moving the vertex v to the point p . We denote the number of crossings in a drawing Γ by $\text{cr}(\Gamma)$, the number of crossings on edges incident to v by $\text{cr}(\Gamma, v)$, and we refer with $\text{cr}(\Gamma, e, f)$ to the number of crossings on two edges e and f in Γ , i.e., $\text{cr}(\Gamma, e, f) \in \{0, 1\}$ if $e \neq f$. For a point u and a segment e , denote by $\mathcal{VR}(u, e)$ the *visibility region of u and e* , i.e., the set of points $p \in \mathbb{R}^2$ such that the segment up and e do not intersect. Moreover, let $\mathcal{BD}(u, e)$ be the boundary of $\mathcal{VR}(u, e)$. Let $\mathcal{A}(\Gamma, v)$ be the arrangement over all boundaries $\mathcal{BD}(u, e)$ for each neighbor $u \in N(v)$ of v and each edge $e \in E \setminus E(u)$; see Figure 1. The arrangement has the property that two points p and q in a common cell of $\mathcal{A}(\Gamma, v)$ induce the same number of crossings for v , i.e., $\text{cr}(\Gamma[v \mapsto p], v) = \text{cr}(\Gamma[v \mapsto q], v)$ [21]. Thus, the computation of a crossing minimal position p^* reduces to finding a *crossing-minimal region f^** in $\mathcal{A}(\Gamma, v)$.

For our experiments, we used two different compute servers. Both systems ran with an openSUSE Leap 15.0 operating system. All algorithms were compiled with `g++` version 7.3.1 with optimization mode `-O3`. System 1 was used for running time experiments, i.e., for the experiments evaluated in Section 3.1 and in Section 4.2. System 2 is used for the experiments evaluated in Section 4.3.

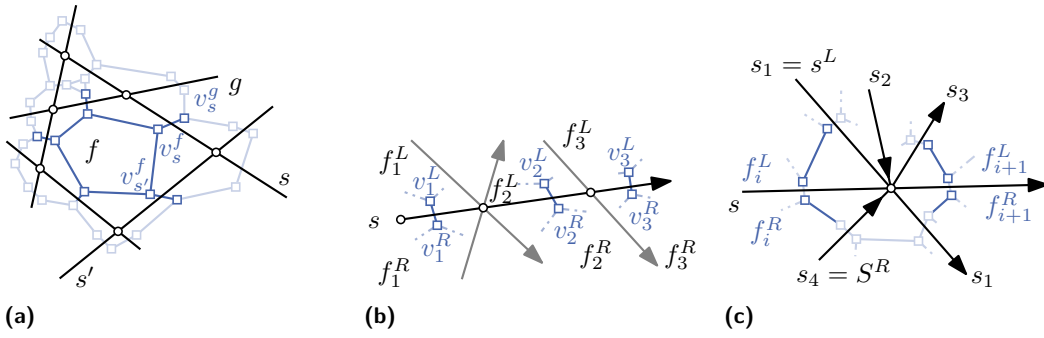
System 1 Intel Xeon(tm) E5-1630v3 processor clocked at 3.7 GHz, 128 GB RAM.

System 2 Two Intel Xeon(tm) E5-2670 CPU processors clocked at 2.6 GHz, 64 GB RAM.

3 Efficient Implementation of the Crossing-Minimal Position

The vertex-movement approach iteratively moves a single vertex to its crossing-minimal position. The running time of the overall algorithm crucially depends on an efficient computation of this operation. Therefore the aim of this section is to provide an efficient implementation of the crossing-minimal position of a vertex. Our previous implementation [21] heavily relies on CGAL [24], which follows an exact computations paradigm and uses exact number types to, e.g., represent coordinates and intermediate results. This helps to ensure correctness but considerably increases the running time of the algorithms. We introduce an approach to compute the crossing-minimal position that drastically reduces the usage of exact computations.

Computing a crossing-minimal position of a vertex v is equivalent to computing a crossing-minimal region f^* in the arrangement $\mathcal{A}(\Gamma, v)$. The region f^* of a vertex v can be computed by a breadth-first search in the dual graph $\mathcal{A}(\Gamma, v)^*$. Thus, the time-consuming steps to



■ **Figure 2** (a) Bloated dual \mathcal{A}^+ (blue) of an arrangement \mathcal{A} (black). Inserting edges dual to a segment s (b) and within a face (c).

compute f^* are to construct the arrangement $\mathcal{A}(\Gamma, v)$ and then to build the dual $\mathcal{A}(\Gamma, v)^*$. Instead of computing the dual $\mathcal{A}(\Gamma, v)^*$ we construct a so-called *bloated dual* $\mathcal{A}(\Gamma, v)^+$. The advantage of this approach is that it suffices to compute the set of intersecting segments in $\mathcal{A}(\Gamma, v)$ to construct $\mathcal{A}(\Gamma, v)^+$ and it is not necessary to compute the arrangement $\mathcal{A}(\Gamma, v)$ itself, i.e., the exact coordinates of each intersection.

Let S be a set of line segments and let \mathcal{A} be the arrangement of S . A *bloated dual* of \mathcal{A} is a graph \mathcal{A}^+ that has the following properties (compare Figure 2a):

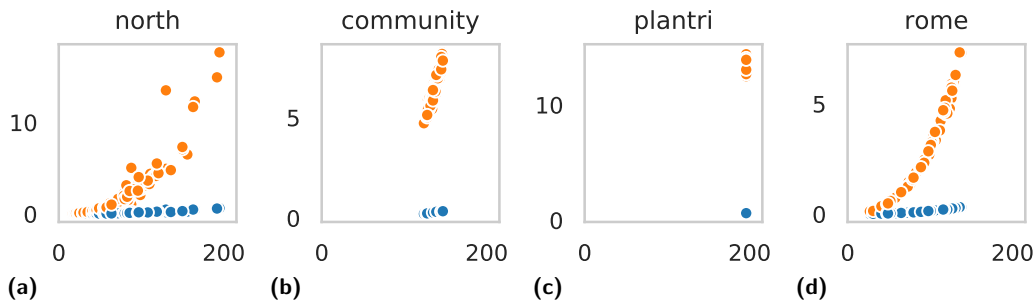
- (i) each edge e incident to a face f corresponds to a vertex v_e^f in \mathcal{A}^+ ,
- (ii) if two distinct segments $s, s' \in S$ of f have a common intersection on the boundary of f , then $v_s^f v_{s'}^f \in E(\mathcal{A}^+)$, and
- (iii) for two distinct faces f, g sharing a common segment s , there is an edge $v_s^f v_s^g \in E(\mathcal{A}^+)$.

Note that given a crossing-minimal face and $v_{s_0}^f$, the geometric representation of f has to be computed in order to compute a crossing-minimal position $p \in f$. Further a vertex $v_{s_0}^f$ belongs to a cycle $v_{s_0}^f, v_{s_1}^f, \dots, v_{s_k}^f$. Then, the geometric representation of the boundary of f can be computed by intersecting the segments s_i and s_{i+1} , where we set $k+1=0$. In the following, we will show that it is sufficient to know the order in which the segments in S intersect to construct the bloated dual. Thus, exact number types only have to be used to determine the order of two segments whose intersections with a third segment s have a small distance on s .

We construct the bloated dual of \mathcal{A} in two steps. First, we insert all vertices v_s^f, v_s^g and the corresponding edge $v_s^f v_s^g$. In the second step, we insert the remaining edges $v_s^f v_{s'}^f$ within a face f . For a compact description we assume that no intersection point of two segments is an endpoint of a segment. We define the *source* of s and *target* of s to be the lexicographically smallest and largest point on s , respectively. We direct each segment s from its source to its target.

Let p_1, p_2, \dots, p_l be the intersection points on a segment s in lexicographical order. These intersection points correspond to a set of left faces $f_1^L, f_2^L, \dots, f_{l+1}^L$ and to a set of right faces $f_1^R, f_2^R, \dots, f_{l+1}^R$, such that f_i^L and f_i^R share parts of their boundary; see Figure 2b. Thus, we can associate a set of vertices $v_i^L, v_i^R, 2 \leq i \leq l+1$, with s , and add the edges $v_i^L v_i^R$ to \mathcal{A}^+ . Note that only the order and not the actual coordinates of the points p_1, \dots, p_l has to be known to insert the edges. Thus, given the set of segments that intersect s , an exact number type is only necessary to determine the order of two segments s_i and s_j whose intersection points p_i and p_j on s have a small distance.

We now add the remaining edges within a face f . Let $S' = \{s_1, \dots, s_k\} \subseteq S$ be the set of segments that intersect s in p_i ; see Figure 2c. The two segments $s^L, s^R \in S'$ that lie on the boundary of f_i^L and f_i^R can be determined as follows. To find the segment s^L , we



■ **Figure 3** Comparing the running time of two approaches (orange PRECISE, blue BD) to compute the crossing minimal region. Each point corresponds to a graph G . The x -axis shows the number of edges of G . The y -axis depicts the running time in seconds to compute the crossing minimal regions for all vertices of G .

distinguish two cases. First, assume that there exists a segment $s' \in S'$ whose source is left of s . Observe that if there is a segment s'' whose target is left of s , the segment s'' cannot be the segment s^L . Thus, we assume without loss of generality that all sources of segments in S'_s are left of s . Then a segment $s' \in S'$ is the segment s^L if and only if the segment s' and each segment $s'' \in S' \setminus \{s'\}$ form a right turn. Now consider the case that there is no segment whose source is left of s . Then a segment s' is s^L if and only if the segment s' and each segment $s'' \in S' \setminus \{s'\}$ form a left turn. The segment s^R can be determined analogously.

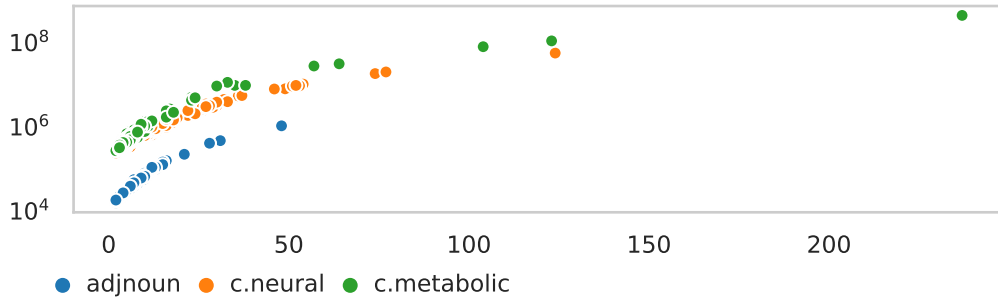
Implementation Details. We give some implementation details which allow us to efficiently implement the construction of the bloated dual. We use the index of a vertex to decide whether it is left or right of s , i.e., vertices with an odd index are left of s and vertices with an even index are right of s . The fact that each vertex of \mathcal{A}^+ has degree at most 3 allows us to represent \mathcal{A}^+ as a single array B of size $3n$, where n is the number of vertices of \mathcal{A}^+ . The vertices incident to a vertex v_i occupy the cells $B[3i]$, $B[3i+1]$ and $B[3i+2]$. Moreover, each pair of segments in S can be handled independently to construct the bloated dual. This enables a parallelization over the segments in S .

3.1 Evaluation of the Running Time

In this section, we compare the running time of the two approaches to compute the crossing-minimal region of a vertex. We refer with PRECISE to the approach that uses CGAL to compute the crossing minimal region and with BD to the approach based on the bloated dual. In order to compute all intersecting segments, we use a naive implementation of a sweep-line algorithm [4]. In this approach all segments within a specific interval are pairwise checked for an intersection. This has the advantage that the computation is independent of the coordinates of the intersection.

The experimental setup is as follows. Given a drawing Γ of a graph G , we are interested in the running time of moving all vertices of a graph to their crossing-minimal positions. Therefore, we measure the running time of computing the crossing-minimal regions of all vertices. In order to guarantee the comparability of the two approaches, we use the same vertex order and only compute the crossing-minimal region but do not update the positions of the vertices. We use the same set of benchmark graphs used in [21]: NORTH¹, ROME¹, graphs

¹ <http://graphdrawing.org/data.html>



■ **Figure 4** The x -axis shows the vertex-degree and the y -axis the number of intersecting edges in the arrangement $\mathcal{A}(\Gamma, v)$. The y -axis is in log-scale.

that have COMMUNITY structure, and TRIANGULATIONS on 64 vertices with an additional 10 random edges. For each graph class, 100 graphs were selected uniformly at random. We use the implementation of STRESS [12] provided by OGDF [7] (snapshot 2017-07-23) to compute an initial layout of the graphs.

The plots in Figure 3 shows the results of the experiments. Each point in the plot corresponds to the running time of computing all crossing-minimal region of a single graph. The plot shows that the BD implementation is considerably faster than the PRECISE implementation. For each graph class, we achieve on average a speed-up of at least 20. The minimum speed-up on the NORTH graphs is 8. For each graph class, the speed-up is at least 18 for at least 75 out of 100 instances.

4 Random Sampling

The worst-case running time of computing the crossing-minimal region of a vertex v is super-quadratic in the size of the graph, see Theorem 1. Figure 4 shows the number of intersecting segment in the arrangement $\mathcal{A}(\Gamma, v)$ compared to the vertex-degree of v , for vertices of three selected graphs with at most 2 133 edges, compare Table 1. For these graphs the arrangement already contains up to 440 685 519 intersecting segments. Indeed, we were not able to compute the number of intersections for all vertices of the graph c.metabolic, since the algorithm ran out of memory first. Due to the high number of intersections in graphs with a high number of edges or a large maximum vertex-degree, it is for these graphs infeasible to compute a crossing-minimal position of a vertex. This motivates the following question: Is a small subgraph of G sufficient to considerably reduce the number of crossings in a given drawing?

To address this question, we follow the *vertex-movement approach*. Let Γ_0 be a drawing of G and let v_1, v_2, \dots, v_n be an ordered set of the vertices V of G . For each vertex v_i we obtain a new drawing Γ_i from the drawing Γ_{i-1} by moving v_i to a new position p_i^* . To compute the new position we consider a *primal* sampling approach, i.e., a sampling of points in the solution space \mathbb{R}^2 , and a *dual* sampling approach, i.e., a sampling of edges that induce constraints to the solution space.

More formally, we consider the following approach to compute a new position of a single vertex v_i . Let $S_i \subset E$ be a uniform random subset of the edges of G and let $V(S_i) \subset V$ be the vertices that are incident to an edge in S_i . The graph $G|_{S_i} = (V(S_i) \cup N(v_i) \cup \{v_i\}, S_i \cup E(v_i))$ induces a drawing $\Gamma|_{S_i}$ in Γ_{i-1} . Let R_i be the crossing-minimal region of v_i with respect to the drawing $\Gamma|_{S_i}$. Recall that for $S_i = E$ the region R_i has the property that $\text{cr}(\Gamma|_{S_i}[v_i \mapsto$

$p], v_i) = \text{cr}(\Gamma|_{S_i}[v_i \mapsto q], v_i)$ for any two points $p, q \in R_i$, compare Section 2. If S_i is a strict subset of E , then R_i does not necessarily have this property anymore. For this reason, let $P_i \subset R_i$ be a set of uniform random points and let $p_i^* \in P_i \cup \{p_i'\}$ be the point that minimizes $\text{cr}(\Gamma[v \mapsto p_i^*], v_i)$, where p_i' is the position of v_i in Γ_{i-1} .

This remainder of this section is organized as follows. First, we analyze the dual sampling from a theoretical perspective (Section 4.1), followed by an experimental evaluation that compares the primal to the dual sampling (Section 4.2). Finally, based on the insights from this evaluation, we introduce in Section 4.3 a *weighted* sampling approach that is less restrictive than the dual sampling.

4.1 Approximating the Co-Crossing Number of a Vertex

In this section we study the dual sampling approach, i.e., the sampling of edges, with tools introduced in the context of the theory of VC-dimension. A thorough introduction into the theory of VC-dimension can be found in Matoušek's *Lectures on Discrete Geometry* [18]. For a fixed vertex v , a drawing Γ is ε -well behaved if for each point $p \in \mathbb{R}^2$ and each vertex $u \in N(v)$, the edge uv crosses at most $(1 - \varepsilon)|E|$ edges in the drawing $\Gamma[v \mapsto p]$. The *co-crossing number* $\text{co-cr}(\Gamma, v)$ of a vertex v is the number of edge pairs $e \in E$ and $uv \in E$ that do not cross. We show that given an ε -well-behaved drawing Γ of a graph $G = (V, E)$ and a degree- k vertex v , a random sample $S \subset E$ of size $\Theta(k \log k)$ enables us to compute a position q^* whose co-crossing number is a $(1 - \delta)$ -approximation of the co-crossing number of a vertex v . Note that we are not able to guarantee that a large co-crossing number of a vertex v implies a small crossing number of v . On the other hand, the co-crossing number is of interest for a variety of (sparse) graph. For example, drawings that contain many triangles are ε -well-behaved, since every line intersects at most two segments of a triangle.

A *set system* is a tuple (X, \mathcal{F}) with a base set X and $\mathcal{F} \subseteq 2^X$. In the following, we assume X to be finite. For some parameters $\varepsilon, \delta \in (0, 1]$, a set $S \subseteq X$ is a *relative (ε, δ) -approximation* for the set system (X, \mathcal{F}) if for each $R \in \mathcal{F}$ the following inequality holds.

$$\left| \frac{|S \cap R|}{|S|} - \frac{|R|}{|X|} \right| \leq \delta \max\left\{ \frac{|R|}{|X|}, \varepsilon \right\} \quad (1)$$

The proof of the following proposition and of proofs of statements that are marked with (\star) can be found in the appendix of the full version.

► **Proposition 2** (\star) . For $\varepsilon, \delta \in (0, 1]$, let S be an (ε, δ) -approximation of the set system (X, \mathcal{F}) . If every $R \in \mathcal{F}$ has size at least $\varepsilon|X|$ then Equation (1) can be rewritten as follows:

$$(1 - \delta)|R| \leq |X| \frac{|S \cap R|}{|S|} \leq (1 + \delta)|R|.$$

Let $\mathcal{F}|_A = \{R \cap A \mid R \in \mathcal{F}\}$ be the restriction of \mathcal{F} to a set $A \subseteq X$. A set $A \subseteq X$ is *shattered* by \mathcal{F} if every subset of A can be obtained by an intersection of A with a set $R \in \mathcal{F}$, i.e., $\mathcal{F}|_A = 2^A$. The *VC-dimension* of a set system (X, \mathcal{F}) is the size of the largest subset $A \subseteq X$ such that A is shattered by \mathcal{F} [25].

► **Theorem 3** (Har-Peled and Sharir [15], Li et al. [17]). Let (X, \mathcal{F}) be a finite set system with VC-dimension d , and let $\delta, \varepsilon, \gamma \in (0, 1]$. A uniform random sample $S \subseteq X$ of size

$$\Theta\left(\frac{d \cdot \log \varepsilon^{-1} + \log \gamma^{-1}}{\varepsilon \delta^2}\right)$$

is a relative (ε, δ) -approximation for (X, \mathcal{F}) with probability $(1 - \gamma)$.

For a vertex $u \in N(v)$, let $\overline{\mathbb{E}_{uv}(\Gamma)} = \{e \in E \mid \text{cr}(\Gamma, e, uv) = 0\}$ denote the set of edges that are not crossed by the edge uv in Γ . Then we have $\text{co-cr}(\Gamma, v) = \sum_{u \in N(v)} |\overline{\mathbb{E}_{uv}(\Gamma)}|$. Moreover, let $\overline{\mathbb{E}_{uv}(p)} = \overline{\mathbb{E}_{uv}(\Gamma[v \mapsto p])}$. Then the set $\overline{\mathcal{F}_{uv}} = \bigcup_{p \in \mathbb{R}^2} \{\overline{\mathbb{E}_{uv}(p)}\}$ contains for each drawing $\Gamma[v \mapsto p]$ the set of edges that are not crossed by the edges uv , i.e., $\overline{\mathbb{E}_{uv}(p)}$. In particular $(E, \overline{\mathcal{F}_{uv}})$ is a set system and we will prove that it has bounded VC-dimension. This allows us to approximate the number of edges that are not crossed by the edge uv . We facilitate this to approximate the co-crossing number of a vertex for ε -well behaved drawings.

► **Lemma 4.** *The VC-dimension of the set system $(E, \overline{\mathcal{F}_{uv}})$ is at most 8.*

Proof. Recall that that vertex u has a fixed position. Let $\mathcal{BD}(u, e)$ be the boundary of the visibility region of u and the edge $e \in E$. Let \mathcal{A} denote the arrangement of all boundaries $\mathcal{BD}(u, e), e \in E$. Let F be the set of faces in \mathcal{A} . Note that by Lemma 3.1 in [21] for every two points $p, q \in f$ the sets E_p and E_q of edges that have a non-empty intersection with the edge uv when v is moved to p and q , respectively, coincide. Hence, the set $E_f \subseteq E$ of edges that cross the edge uv , in the drawing obtained from Γ where v is moved to an arbitrary position in f , is well defined. Thus, the number of faces $|F|$ is an upper bound for $|\overline{\mathcal{F}_{uv}|_A}$ for every $A \subseteq E$. Note that there may be subsets of E that are represented by more than one face. Moreover, observe that the visibility region $\mathcal{VR}(u, e)$ is the intersection of three half-planes. Let l_e^1, l_e^2, l_e^3 be the supporting lines of these half-planes and let \mathcal{A}' be the arrangement of lines $l_e^i, e \in E$. Hence, the number of faces in the arrangement \mathcal{A}' of $3m$ lines is an upper bound for $|F|$, with $m = |E|$. The number of faces $|F'|$ of \mathcal{A}' is bounded by $f(m) := 3m(3m - 1)/2 + 1$ [20]. Thus, it is not possible to shatter a set $A \subseteq E$ if the number of faces $|F'|$ is smaller than the number of subsets of A . The largest number for which the equality $2^m \leq f(m)$ holds is between 8 and 9. Since 2^m grows faster than $f(m)$, the largest set that can possibly be shattered has size at most 8. ◀

Due to Proposition 2 and Theorem 3 a relative (ε, δ) -approximation S_u of $(E, \overline{\mathcal{F}_{uv}})$ allows us to approximate the number of edges that are not crossed by the edge uv . In the following we show that we can approximate the co-crossing number of a vertex v in any drawing $\Gamma[v \mapsto p]$ if we are given a relative (ε, δ) -approximation S_u for each vertex u that is adjacent to v . The number $|\overline{\mathbb{E}_{uv}(p)} \cap S_u|/|S_u|$ corresponds to the relative number of edges in S_u that are not crossed by the edge uv . Hence, the function $\lambda(p) = |E| \sum_{u \in U} |\overline{\mathbb{E}_{uv}(p)} \cap S_u|/|S_u|$ can be seen as an estimation of $\text{co-cr}(p) = \text{co-cr}(\Gamma[v \mapsto p], v)$.

► **Lemma 5** (\star). *Let $\varepsilon, \delta \in (0, 1]$ be two parameters and let Γ be an ε -well behaved drawing of G . For every $u \in N(v)$, let S_u be a relative (ε, δ) -approximation of the set system $(E, \overline{\mathcal{F}_{uv}})$. Then $(1 - \delta) \text{co-cr}(p) \leq \lambda(p) \leq (1 + \delta) \text{co-cr}(p)$ holds for all $p \in \mathbb{R}^2$.*

Assume that $\varepsilon, \delta, \gamma \in (0, 1)$ are constants. Lemma 5 shows that k independent samples S_u of constant size approximate the co-crossing number of v . By slightly increasing the number of samples, we can use a single set S for all neighbors u . This reduces the running time from $O(k^3 \log k)$ to $O(k^2 \log^3 k)$.

► **Lemma 6** (\star). *Let v be a degree- k vertex and let $\varepsilon, \delta, \gamma \in (0, 1]$ with $\gamma \leq 1/k$. A uniformly random sample $S \subseteq E$ of size $\Theta((\log \varepsilon^{-1} + \log \gamma^{-1})/(\varepsilon \delta^2))$ is a relative (ε, δ) -approximation of the set system $(E, \overline{\mathcal{F}_{uv}})$ with probability $1 - k\gamma$, for each $uv \in E$.*

With Lemma 5 and Lemma 6 at hand, we have all the necessary tools to prove the main theorem.

► **Theorem 7.** *Let $\varepsilon, \delta, \gamma \in (0, 1]$ be three constants and let $G = (V, E)$ be a graph with a ε -well behaved drawing Γ and let $v \in V$ be a degree- k vertex. Let p^* be the position that maximizes $\text{co-cr}(\Gamma[v \mapsto p^*], v)$. A $(1 - \delta)$ -approximation of $\text{co-cr}(\Gamma[v \mapsto p^*])$ can be computed in $O(k^2 \log^3 k)$ time with probability $1 - \gamma$.*

Proof. Let $\gamma' = \gamma \cdot k^{-1}$ and $\delta' = \delta/2$. Let $S \subseteq E$ be a uniformly random sample of size $\Theta((\log \varepsilon^{-1} + \log \gamma'^{-1})/(\varepsilon \delta'^2))$. According to Lemma 6, for each $uv \in E$, the sample S is a (ε, δ') -approximation of the $(E, \overline{\mathcal{F}}_{uv})$ with probability $1 - k\gamma' = 1 - \gamma$.

According to Lemma 5 the expected number of crossing-free edges $\lambda(p)$ is a $(1 - \delta)$ -approximation of $\text{co-cr}(p)$, i.e., $(1 + \delta') \text{co-cr}(q) \geq \lambda(q) \geq (1 - \delta') \text{co-cr}(q)$. Let p^* be the position that maximizes $\text{co-cr}(p)$ and let q^* be the position that maximizes $\lambda(q)$. Hence, we have $\lambda(q^*) \geq \lambda(p^*)$. Observe that over $\delta' > 0$ the inequality $(1 - \delta')/(1 + \delta') \geq 1 - 2\delta'$ holds. We use this to prove that $\text{co-cr}(q^*) \geq (1 - 2\delta') \text{co-cr}(p^*)$.

$$\text{co-cr}(q^*) \geq \frac{1}{(1 + \delta')} \lambda(q^*) \geq \frac{1}{(1 + \delta')} \lambda(p^*) \geq \frac{1 - \delta'}{1 + \delta'} \text{co-cr}(p^*) \geq (1 - 2\delta') \text{co-cr}(p^*)$$

Plugging in the value $\delta/2$ for δ' yields that $\text{co-cr}(q^*)$ is a δ -approximation of $\text{co-cr}(p^*)$. Since the three parameters $\varepsilon, \delta, \gamma$ are constants, the size of the sample S is in $\Theta(\log k)$. Recall that the running time to compute the crossing-minimal position of v in a drawing Γ is $O((kn + m)^2 \log(kn + m))$ (Theorem 1). Thus the position q^* can be computed in $O(k \log k + \log k)^2 \log(k \log k + \log k)$ time, since $m = |S| \in \Theta(\log k)$ and $n \leq 2m$. The following estimation concludes the proof.

$$O(k^2 \log^2 k \log(k \log k)) = O(k^2 \log^2 k \log(k^2)) = O(k^2 \log^3 k) \quad \blacktriangleleft$$

Note that the previous techniques can be used to design a δ -approximation algorithm for the crossing number of a vertex. But this requires drawings of graphs where at least $\varepsilon|E|$ edges, i.e., $\Omega(|E|)$, are crossed. This restriction is not too surprising, since sampling the set of edges can result in an arbitrarily bad approximation for a vertex whose crossing-minimal position induces no crossings.

4.2 Experimental Evaluation

In this section we complement the theoretical analyses of the random sampling of edges with an experimental evaluation. We first introduce our benchmark instances, followed by a description of a preprocessing step to reduce trivial cases and a set of configurations that we evaluate.

Benchmark Instances. We evaluate our algorithm on graphs from three different sources.

DIMACS The graphs from this classes are selected from the 10th DIMACS Implementation Challenge - Graph Partitioning and Graph Clustering [2].

Sparse MC Inspired by the selection of benchmark graphs in [19], we selected a few arbitrary graphs from the Suite Sparse Matrix Collection (formerly known as the Florida Sparse Matrix Collection) [9].

k -regular For each $k = 3, 6, 9$ we computed 25 random k -regular graphs on 1000 vertices following the model of Steger and Wormald [23].

Preprocessing. Some of the benchmark graphs contain multiple connected components. Moreover, we observed that the STRESS layout introduces crossings with edges that are incident to a degree-1 vertex. In both cases, these crossings can be removed. Therefore, we

reduce the benchmark instances so that they do not contain these trivial cases as follows. First, we evaluate only the connected component G_C of each graph G that has the highest number of vertices. Further, we iteratively remove all vertices of degree 1 from G_C .

The vertex-movement approach takes an initial drawing of a graph as input. Note that the experimental results in [21] showed that drawings obtained with STRESS have the smallest number of crossings compared to other energy-based methods implemented in OGDF. In order to avoid side effects, we first computed a random drawing for each graph G_C where each coordinate is chosen uniformly at random on a grid of size $m \times m$. Afterwards we applied the STRESS method implemented in OGDF [7] (snapshot 2017-07-23) to this drawing.

Configurations. The previously described approach moves the vertices in a certain order. We use the order proposed in [21], i.e, in descending order with respect to the function $\text{cr}(\Gamma_0, v_i)^2, v_i \in V$, where Γ_0 is the initial drawing. The computation of the new position p_i^* of a vertex v_i depends on three parameters $(|S_i|, |P_i|, K)$. The parameter K is a threshold on the degree k_i of v_i , since we observed in our preliminary experiments, that in case that k_i is large, 128 GB of memory are not sufficient to compute the crossing-minimal region. Note that in case that $|S_i|$ is constant the running time to compute R_i is $O((k_i \cdot n')^2 \log n') = O(k_i^2)$, where $n' = |V(S)| \in O(|S|)$. We handle vertices of degree larger than K , as follows. Let $N_1 \cup \dots \cup N_l$ be a partition of the neighborhood $N(v)$ of v with $l = \lfloor |N(v)|/K \rfloor$. Further, let u_1, u_2, \dots, u_k be a random order of $N(v)$, then N_j contains the vertices u_a with $j \leq a \leq j+K$. For each j , we compute a random sample S_i^j and a crossing-minimal position q_j^* of vertex v with neighborhood N_j with respect to S_i^j . The new position p_i^* of v_i is the position that minimizes $\text{cr}(\Gamma[v_i \mapsto q_j^*], v_i)$.

We select the same parameters for each vertex and thus denote the triple by $(|S|, |P|, K)$. We expect that with an increasing number $|S|$ the number of crossings decreases. The sample size $|S| = 512$, was the largest number of samples such that we are able to compute a final drawing of our benchmark instances in reasonable time. As a baseline we sample 1000 points in the plane. Thus, we evaluate the following two configuration, $\mathcal{S}_{512} = (512, 1, 100)$ and $\mathcal{S}_0 = (0, 1000, \infty)$. Finally, we restrict the movement of a single vertex to be within an axis-aligned square that is twice the size of the smallest axis-aligned squares that entirely contains Γ_0 .

Evaluation. Table 1 lists statistics for the DIMACS and the SPARSE MC graphs. In particular the number of crossings of the initial drawing (STRESS) and the drawing obtained by the \mathcal{S}_{512} and \mathcal{S}_0 configurations. Furthermore, we report the running times for the two configurations. Since we use an external library (OGDF) to compute the initial drawing, the reported times do not include the time to compute the initial drawing. Note that STRESS required at most 0.9 min to complete on the DIMACS graph and 2.3 min on the SPARSE MC graphs. Since the size of the arrangement $\mathcal{A}(\Gamma, v)$ depends on the degree of v , the overall running time varies with the number of vertices and the average degree. Compare, e.g., c.metabolic to c.neural, or mk9-b2 to bcsstk08. Moreover, the commanche graph shows that the running time of \mathcal{S}_0 is not necessarily smaller than the running time of \mathcal{S}_{512} . For each point $p \in P$ the number of crossings of edges incident to v in $\Gamma[v \mapsto p]$ have to be counted. Since the commanche graph contains over 11 000 edges, the \mathcal{S}_{512} configuration with $|P| = 1$ is faster than the \mathcal{S}_0 configuration, which has to count the number of crossings for 1 000 points.

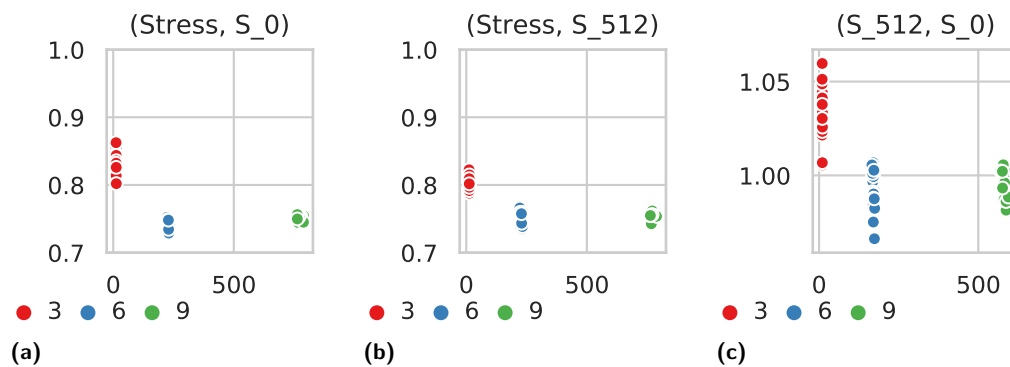
Now consider the number of crossings in the initial drawing (STRESS) and in the drawing obtained by the \mathcal{S}_{512} configuration. Since we move a vertex only if it decreases its number of crossings, it is expected that the number of crossings decreases on all graphs. For most

■ **Table 1** Statistics for the DIMACS and SPARSE MC graphs. n , m , and $\bar{\Delta}$ correspond the number of vertices, edges and the mean vertex-degree, respectively.

	n	m	$\bar{\Delta}$	crossings			time [min]	
				STRESS	\mathcal{S}_{512}	\mathcal{S}_0	\mathcal{S}_{512}	\mathcal{S}_0
DIMACS								
adjnoun	102	415	8.14	6 576	3 775	4 468	0.11	0.09
football	115	613	10.66	6 865	3 568	4 030	0.14	0.17
netscience	352	887	5.04	1 724	583	814	0.53	0.31
c.metabolic	445	2 017	9.07	113 117	55 714	63 028	11.29	2.29
c.neural	282	2 133	15.13	128 068	86 641	90 920	5.23	2.07
jazz	193	2 737	28.36	223 990	143 647	153 040	5.22	3.31
power	3 353	5 006	2.99	7 622	6 854	6 293	4.56	10.74
email	978	5 296	10.83	504 144	342 020	357 272	37.12	12.48
hep-th	4 786	12 766	5.33	836 809	546 780	638 069	72.86	78.24
SPARSE MC								
1138_bus	671	991	2.95	657	402	467	0.41	0.33
ch7-6-b1	630	1 243	3.95	64 055	24 928	26 055	6.54	0.79
mk9-b2	1 260	3 774	5.99	412 397	248 884	252 198	20.33	7.14
bcsstk08	1 055	5 927	11.24	455 069	342 996	344 644	67.30	18.70
mahindas	1 258	7 513	11.94	1 463 437	933 247	1 042 787	68.17	24.09
eris1176	892	8 405	18.85	1 682 458	1 030 881	1 087 605	77.09	27.33
commanche	7 920	11 880	3.00	6 332	6 239	6 146	6.52	56.75

graphs, the \mathcal{S}_{512} configuration decreases the number of crossings by over 30%. In case of the ch7-6-b1 and the netscience graph the number of crossings are even decreased by over 60%. Exceptions are the bcsstk08, power and commanche graphs with 24%, 10% and 1.4% respectively. Comparing the number crossings obtained by \mathcal{S}_{512} to the configuration \mathcal{S}_0 , \mathcal{S}_0 results in fewer crossings only on two graphs (power, commanche).

Observe that the power, 11138_bus, ch7-6-b1 and commanche graphs all have an average vertex-degree of roughly 3.0. The comparison of the number of crossing obtained by \mathcal{S}_{512} and \mathcal{S}_0 is not conclusive, since \mathcal{S}_0 yields fewer crossings on the power and commanche graphs and \mathcal{S}_{512} on the remaining two. In order to be able to further study the effect of the (average) vertex degree we evaluate the number of crossings of k -regular graphs. We use the plots



■ **Figure 5** Number of crossings of the k -regular graphs.

in Figure 5 for the evaluation. Each point (x_G, y_G) corresponds to a k -regular graph G . The color encodes the vertex-degree. Let Γ_A and Γ_B be two drawings of G obtained by an algorithm A and B , respectively. The x -value x_G corresponds to the number of crossings in Γ_A in thousands, i.e., $\text{cr}(\Gamma_A)/1000$. The y -value y_G is the quotient $\text{cr}(\Gamma_B)/\text{cr}(\Gamma_A)$. The titles of the plots are in the form (A, B) and encode the compared algorithms. For example in Figure 5a algorithm A is STRESS and B is \mathcal{S}_0 . For example, the STRESS drawings of the 3-regular graphs have on average 12 487 crossings. Drawings obtained by \mathcal{S}_0 have on average 17% less crossings, i.e., 10 402. On the other hand, \mathcal{S}_{512} decreases the number of crossings on average by 20%. For $k = 6, 9$, \mathcal{S}_0 and \mathcal{S}_{512} both reduce the number of crossings by 25%. In particular, Figure 5c shows that for $k = 6, 9$ it is unclear, whether \mathcal{S}_{512} or \mathcal{S}_0 computes drawings with fewer crossings.

4.3 Weighted Sampling

For some graphs, the previous section gives first indications that sampling a set of edges yields a small number of crossings compared to a pure sampling of points in the plane. In particular Figure 5c indicates that the edge-sampling approach does not always have a clear advantage over sampling points in the plane. One reason for this might be that sampling within the set of points P_i in the region R_i is too restrictive. Observe that the region R_i is only crossing-minimal with respect to the sample S and does not necessarily contain the crossing-minimal position p_i^* of the vertex v_i with respect to all edges E . On the other hand, sampling the set of points P_i in \mathbb{R}^2 does not use the structure of the graph at all. This motivates the following *weighted* approach of sampling points in \mathbb{R}^2 .

For a set $S \subset E$, let cr_j be the number of crossings of the vertex v_i with respect to $\Gamma|_S$, when v_i is moved to a cell c_j of the arrangement $\mathcal{A}(\Gamma|_S, v_i)$. Let M be the maximum of all cr_j . We select a cell c_j with the probability $2^{M-\text{cr}_j} / \sum_k 2^{M-\text{cr}_k}$. Within a given cell, we draw a point uniformly at random. Note that in case that there are exactly n cells such that cell c_j induces j crossings, the probability that the cell c_0 is drawn converges to $1/2$ for $n \rightarrow \infty$.

Benchmark Instances, Preprocessing & Methodology. We use the same benchmark set and the same preprocessing steps as described in Section 4. In order to obtain more reliable results, we perform 10 independent iterations for each configuration on the DIMACS and SPARSE MC graphs. Since the k -regular graphs are uniform randomly computed, they are already representative for their class. Therefore, we perform only single runs on these graphs.

Configuration. We compare the following three configurations. \mathcal{R}_0 refers to the uniform random sampling of points in \mathbb{R}^2 with the parameters $(|S|, |P|, K) = (0, 1000, \infty)$, \mathcal{R}_{512} to the restricted sampling in R_i with the parameters, $(512, 1000, 100)$, and \mathcal{W}_{512} to the weighted sampling in \mathbb{R}^2 with the parameters $(512, 1000, 100)$. The configurations are selected such that \mathcal{R}_0 and \mathcal{R}_{512} differ only in a single parameter, i.e., in the number of sampled edges. The only difference between \mathcal{R}_{512} and \mathcal{W}_{512} is the sampling strategy. Note that the parameters of \mathcal{R}_0 and \mathcal{S}_0 coincide, but not the parameters of \mathcal{S}_{512} and \mathcal{R}_{512} .

Evaluation. Since we executed 10 independent runs of the algorithm on each graph, Table 2 lists the mean and standard deviation of the computed number of crossings for each graph. For each graph, we marked the cell with the lowest number of crossings in green and the largest number in blue. For each graph, we used the Mann-Witney-U test [22] to check the null hypothesis that the crossing numbers belong to the same distribution. The test indicates

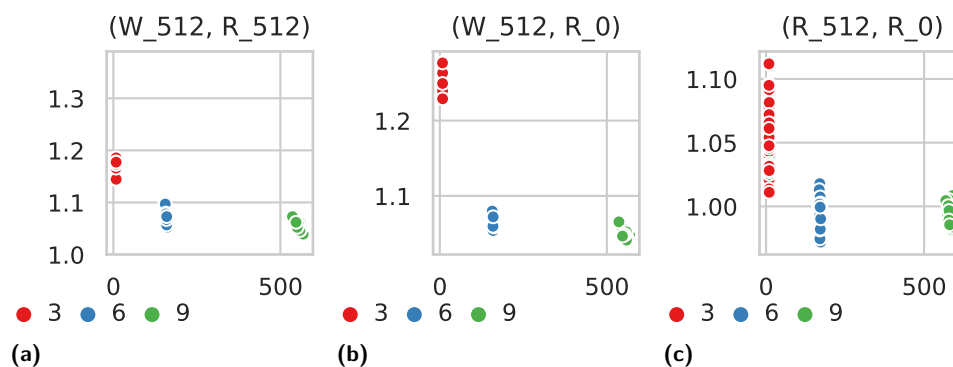
■ **Table 2** Mean and standard deviation (std) of the number of crossing categorized by configuration. For each graph the configuration with the lowest and highest number of crossings in marked.

	\mathcal{R}_0		\mathcal{R}_{512}		\mathcal{W}_{512}	
	mean	std	mean	std	mean	std
DIMACS						
adjnoun	4 445.0	39.55	3 655.7	62.96	3 951.2	19.53
football	3 973.6	97.93	3 350.0	83.38	3 247.0	73.84
netscience	819.0	30.73	497.1	28.78	437.8	12.87
c.metabolic	62 170.4	760.47	56 032.3	1 227.23	62 987.9	1 907.64
c.neural	89 744.3	1 239.22	86 500.8	1 364.5	99 426.1	1 258.98
jazz	152 013.8	1 930.13	147 387.1	3 134.15	213 019.4	1 696.07
power	6 301.1	33.51	4 512.8	63.09	3 912.5	30.97
email	356 583.4	3 512.0	341 503.8	3 480.74	351 168.7	2 624.18
hep-th	640 515.2	3 443.22	515 109.1	3 983.23	392 189.7	1 551.53
SPARSE MC						
1138_bus	474.6	13.25	342.9	12.91	247.6	9.8
ch7-6-b1	25 874.7	356.58	25 172.4	582.48	28 443.5	960.3
mk9-b2	251 360.9	1 514.05	245 447.4	2 914.18	228 794.5	2 069.96
bcsstk08	346 404.4	3 730.3	328 182.0	6 127.69	330 213.8	1 726.01
mahindas	1 036 745.7	11 494.88	936 889.0	11 207.34	1 105 850.9	10 185.51
eris1176	1 103 184.6	21 475.11	1 037 509.5	29 877.3	1 492 423.4	25 457.93
commanche	6 135.2	13.08	5 370.3	24.75	5 979.4	14.72

that we can reject the null hypothesis at a significance level of $\alpha = 0.01$, for all graphs with the exception of football, ch7-6-b1 and bcsstk08. First, observe that the \mathcal{R}_0 configuration never computes a drawing with fewer crossings than \mathcal{R}_{512} . Including the football, ch7-6-b1 and the bcsstk08 graphs, 11 of the drawings with the fewest crossing were obtained from the \mathcal{R}_{512} configurations. Only 7 correspond to the \mathcal{W}_{512} configuration. Table 1 shows that these graphs have an average vertex-degree of at most 11. Moreover, the degree-distributions of these graphs follow the power-law (compare full version). On the other hand, a few of the 8 graph where \mathcal{R}_{512} outperforms \mathcal{W}_{512} also have a small average vertex-degree.

We use Figure 6 to compare the effect of the vertex-degree on the number of crossings. The plot follows the same convention as the plots in Figure 5. Observe that for each k , the \mathcal{W}_{512} configuration computes drawings with fewer crossings than \mathcal{R}_{512} . The improvement decreases with an increasing k . The same observation can be made for the comparison of \mathcal{W}_{512} to \mathcal{R}_0 but not for the comparison for \mathcal{R}_{512} to \mathcal{R}_0 , which indicates that sampling the set of points P_i within the region R_i is indeed too restrictive, at least on our k -regular graphs.

Overall our experimental evaluation shows that even with a naive uniform random sampling of a set of points in the plane the number of crossings in drawings of STRESS can be reduced considerably. Using a random sample of a subset of the edges helps to compute drawings with even less crossings. The mean-vertex degree and the degree-distributions are good indicators for whether the restrictive or the weighted sampling of the point set P_i results in a drawing with the smallest number of crossings.



■ **Figure 6** Comparison of the number of crossing of the k -regular graphs computed by \mathcal{W}_{512} and \mathcal{R}_{512} .

5 Conclusion

In our previous work we showed that the primitive operation of moving a single vertex to its crossing-minimal position significantly reduces the number of crossings compared to drawings obtained by STRESS. In this paper we introduced the concept of *bloated dual of line arrangements*, a combinatorial technique to compute a dual representation of line arrangements. In our applications of computing drawings with a small number of crossings, this technique resulted in a speed-up of factor of 20. This improvement was necessary to adapt the approach for graphs with a large number of vertices and edges. On the other hand, since the worst-case running time is super-quadratic, this improvement is not sufficient to cope with large graphs. In Section 4 we showed that random sampling is a promising technique to minimize crossings in geometric drawings. In Section 4.1 we proved that a random subset of edges of size $\Theta(k \log k)$ approximates the co-crossing number of a vertex v with a high probability. Further, we evaluated three different strategies to sample a set of points in the plane in order to compute a new position for the vertex v_i . First, the evaluation confirms that the number of crossings compared to STRESS can be reduced considerably. Furthermore, sampling a small subset of the edges is sufficient to reduce the number of crossings compared to a naive sampling of points the plane. Our evaluation suggests that weighted sampling is a promising approach to reduce the number of crossings in graphs with a low average vertex degree. Otherwise, the evaluation indicates that restricted sampling results in fewer crossings.

The running time of the vertex-movement approach in combination with the sampling of the edges mostly depends on the number of vertices. Since a single movement of a vertex is not optimal anymore, two vertices can be moved independently. Thus, future research should be concerned with the question whether a parallelization over the vertex set is able to further reduce the running time while preserving the small number of crossings. Moreover, we ask whether it is sufficient to move a small subset of the vertices to considerably reduce the number of crossings.

References


- 1 Oswin Aichholzer. On the Rectilinear Crossing Number. (<http://www.ist.tugraz.at/staff/aichholzer/research/rp/triangulations/crossing>), May 2017.
- 2 David A. Bader, Andrea Kappes, Henning Meyerhenke, Peter Sanders, Christian Schulz, and Dorothea Wagner. Benchmarking for Graph Clustering and Partitioning. In *Encyclopedia of Social Network Analysis and Mining, 2nd Edition*. Springer-Verlag, 2018. doi:10.1007/978-1-4939-7131-2_23.

- 3 Michael A. Bekos, Henry Förster, Christian Geckeler, Lukas Holländer, Michael Kaufmann, Amadäus M. Spallek, and Jan Splett. A Heuristic Approach Towards Drawings of Graphs with High Crossing Resolution. In *Proceedings of the 26th International Symposium on Graph Drawing (GD'18)*, volume 11282 of *Lecture Notes in Computer Science*, pages 271–285, 2018. doi:10.1007/978-3-030-04414-5_19.
- 4 John L. Bentley and Thomas A. Ottmann. Algorithms for Reporting and Counting Geometric Intersections. *IEEE Transactions on Computers*, C-28(9):643–647, 1979.
- 5 Daniel Bienstock. Some Provably Hard Crossing Number Problems. *Discrete & Computational Geometry*, 6(1):443–459, 1991. doi:10.1007/BF02574701.
- 6 Christoph Buchheim, Markus Chimani, Carsten Gutwenger, Michael Jünger, and Petra Mutzel. Crossings and Planarization. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*, chapter 2, pages 43–85. Chapman and Hall/CRC, 2013.
- 7 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 of Graph Drawing and Visualization*, chapter 17, pages 543–569. Chapman and Hall/CRC, 2013.
- 8 Markus Chimani and Petr Hlinený. Inserting Multiple Edges into a Planar Graph. In Sándor Fekete and Anna Lubiw, editors, *Proceedings of the 32nd Annual Symposium on Computational Geometry (SoCG'16)*, volume 51 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.SocG.2016.30.
- 9 Timothy A. Davis and Yifan Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1):1:1–1:25, 2011. doi:10.1145/2049662.2049663.
- 10 Almut Demel, Dominik Dürrschnabel, Tamara Mchedlidze, Marcel Radermacher, and Lasse Wulf. A Greedy Heuristic for Crossing-Angle Maximization. In *Proceedings of the 26th International Symposium on Graph Drawing (GD'18)*, *Lecture Notes in Computer Science*, pages 286–299, 2018. doi:10.1007/978-3-030-04414-5_20.
- 11 Ruy Fabila-Monroy and Jorge López. Computational Search of Small Point Sets with Small Rectilinear Crossing Number. *Journal of Graph Algorithms and Applications*, 18(3):393–399, 2014. doi:10.7155/jgaa.00328.
- 12 Emden R. Gansner, Yehuda Koren, and Stephen North. Graph Drawing by Stress Majorization. In János Pach, editor, *Proceedings of the 12th International Symposium on Graph Drawing (GD'04)*, volume 3383 of *Lecture Notes in Computer Science*, pages 239–250. Springer Berlin/Heidelberg, 2005. doi:10.1007/978-3-540-31843-9_25.
- 13 Michael R. Garey and David S. Johnson. Crossing Number is NP-Complete. *SIAM Journal on Algebraic and Discrete Methods*, 4(3):312–316, 1983.
- 14 Carsten Gutwenger, Petra Mutzel, and René Weiskircher. Inserting an Edge into a Planar Graph. *Algorithmica*, 41(4):289–308, 2005. doi:10.1007/s00453-004-1128-8.
- 15 Sariel Har-Peled and Micha Sharir. Relative (p, ϵ) -Approximations in Geometry. *Discrete & Computational Geometry*, 45(3):462–496, 2011. doi:10.1007/s00454-010-9248-1.
- 16 Stephen G. Kobourov. Force-Directed Drawing Algorithms. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*, chapter 12. Chapman and Hall/CRC, 2013.
- 17 Yi Li, Philip M. Long, and Aravind Srinivasan. Improved Bounds on the Sample Complexity of Learning. *Journal of Computer and System Sciences*, 62(3):516–527, 2001. doi:10.1006/jcss.2000.1741.
- 18 Jiří Matoušek. *Lectures on Discrete Geometry*, volume 212. Springer New York, 2002.
- 19 Henning Meyerhenke, Martin Nöllenburg, and Christian Schulz. Drawing Large Graphs by Multilevel Maxent-Stress Optimization. *IEEE Transactions on Visualization and Computer Graphics*, 24(5):1814–1827, 2018. doi:10.1109/TVCG.2017.2689016.
- 20 Thomas L. Moore. Using Euler's Formula to Solve Plane Separation Problems. *The College Mathematics Journal*, 22(2):125–130, 1991.

76:16 Geometric Crossing Minimization

- 21 Marcel Radermacher, Klara Reichard, Ignaz Rutter, and Dorothea Wagner. A Geometric Heuristic for Rectilinear Crossing Minimization. In *Proceedings of the 20th Workshop on Algorithm Engineering and Experiments (ALENEX'18)*, pages 129–138, 2018. doi:10.1137/1.9781611975055.12.
- 22 David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman and Hall/CRC, 2003.
- 23 Angelika Steger and Nicholas C. Wormald. Generating Random Regular Graphs Quickly. *Combinatorics, Probability and Computing*, 8(4):377–396, 1999.
- 24 The CGAL Project. *CGAL User and Reference Manual (<http://doc.cgal.org/4.10/Manual/packages.html>)*. CGAL Editorial Board, 4.10 edition, 2017.
- 25 Vladimir N. Vapnik and Alexey Y. Chervonenkis. On the Uniform Convergence of Relative Frequencies of Event to their Probabilities. *Theory of Probability & Its Application*, 16(2):264–280, 1971.
- 26 Imrich Vrt'o. Bibliography on Crossing Numbers of Graphs. (<ftp://ftp.ifi.savba.sk/pub/imrich/crobib.pdf>), 2014.

An Approximate Kernel for Connected Feedback Vertex Set

M. S. Ramanujan 

University of Warwick, UK

https://warwick.ac.uk/fac/sci/dcs/people/ramanujan_sridharan/

R.Maadapuzhi-Sridharan@warwick.ac.uk

Abstract

The FEEDBACK VERTEX SET problem is a fundamental computational problem which has been the subject of intensive study in various domains of algorithmics. In this problem, one is given an undirected graph G and an integer k as input. The objective is to determine whether at most k vertices can be deleted from G such that the resulting graph is acyclic. The study of preprocessing algorithms for this problem has a long and rich history, culminating in the quadratic kernelization of Thomasse [SODA 2010].

However, it is known that when the solution is required to induce a *connected* subgraph (such a set is called a connected feedback vertex set), a polynomial kernelization is unlikely to exist and the problem is NP-hard to approximate below a factor of 2 (assuming the Unique Games Conjecture).

In this paper, we show that if one is interested in only preserving *approximate* solutions (even of quality arbitrarily close to the optimum), then there is a drastic improvement in our ability to preprocess this problem. Specifically, we prove that for every fixed $0 < \varepsilon < 1$, graph G , and $k \in \mathbb{N}$, the following holds.

There is a polynomial time computable graph G' of size $k^{\mathcal{O}(1)}$ such that for every $c \geq 1$, any c -approximate connected feedback vertex set of G' of size at most k is a $c \cdot (1 + \varepsilon)$ -approximate connected feedback vertex set of G .

Our result adds to the set of approximate kernelization algorithms introduced by Lokshtanov et al. [STOC 2017]. As a consequence of our main result, we show that CONNECTED FEEDBACK VERTEX SET can be approximated within a factor $\min\{\text{OPT}^{\mathcal{O}(1)}, n^{1-\delta}\}$ in polynomial time for some $\delta > 0$.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Mathematics of computing → Approximation algorithms

Keywords and phrases Parameterized Complexity, Kernelization, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.77

1 Introduction

Polynomial time preprocessing is one of the widely used methods to tackle NP-hardness in practice, and the area of *kernelization* has been extremely successful in laying down a mathematical framework for the design and rigorous analysis of preprocessing algorithms for decision problems. The central notion in kernelization is that of a *kernelization algorithm*, which is a preprocessing algorithm that runs in polynomial time and transforms a “large” instance of a decision problem into a significantly smaller, but equivalent instance (called a *kernel*). Over the last decade, the area of kernelization has seen the development of a wide range of tools to design preprocessing algorithms, as well as a rich theory of lower bounds based on assumptions from complexity theory [2, 6, 3, 15, 5, 9, 18, 7, 17]. We refer the reader to the survey articles by Kratsch [19] or Lokshtanov et al. [20] for relatively recent developments, or the textbooks [4, 8], for an introduction to the field.



© M. S. Ramanujan;

licensed under Creative Commons License CC-BY

27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 77; pp. 77:1–77:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

An “efficient preprocessing algorithm” in this setting is referred to as a *polynomial kernelization* and is simply a kernelization whose output has size bounded polynomially in a parameter of the input. The central classification task in the area is to classify any problem as one which has a polynomial kernel, or as one that does not.

One fundamental class of problems for which polynomial kernels have been ruled out under certain complexity theoretic hypotheses, is the class of “subgraph hitting” problems *with a connectivity constraint*. It is well-known that placing connectivity constraints on certain subgraph hitting problems can have a dramatic effect on their amenability to preprocessing. A case in point is the classic VERTEX COVER problem. This problem is known to admit a kernelization whose output has $\mathcal{O}(k)$ vertices [4]. However, the CONNECTED VERTEX COVER problem is amongst the earliest problems shown to exclude a polynomial kernel [7] (under a complexity theoretic hypothesis) and this lower bound immediately rules out the possibility of such a kernelization for numerous well-studied generalizations of it. Consequently, obtaining a finer understanding of the impact of connectivity constraints on the limits of preprocessing is an important objective in furthering the study of preprocessing techniques. This is even more relevant when one intends to run approximation algorithms or heuristics on the preprocessed instance.

Unfortunately, the existing notion of kernels, having been built around decision problems, does not combine well with approximation algorithms and heuristics. In particular, in order for kernels to be useful, one is required to solve the preprocessed instance *exactly*. However, this may not always be possible and the existing theory of kernelization says nothing about the inference of useful information from a good *approximate* solution for the preprocessed instance. In order to facilitate the rigorous analysis of preprocessing algorithms in conjunction with approximation algorithms, Lokshtanov et al. [22] introduced the notion of α -*approximate kernels*. Informally speaking, an α -approximate kernelization is a polynomial-time algorithm that, given an instance (I, k) of a *parameterized problem*, outputs an instance (I', k') such that $|I'| + k' \leq g(k)$ for some computable function g and any c -approximate solution to the instance (I', k') can be turned in polynomial time into a $(c \cdot \alpha)$ -approximate solution to the original instance (I, k) .

As earlier, the notion of “efficiency” in this context is captured by the function g being polynomially bounded, in which case we call this algorithm, an α -*approximate polynomial kernelization*. We refer the reader to Section 2 for a formal definition of all terms related to (approximate) kernelization.

In their work, Lokshtanov et al. considered several problems which are known to exclude polynomial kernels and presented an α -approximate polynomial kernelization for these problems for *every* fixed $\alpha > 1$. This implies that allowing for an arbitrarily small amount of error while preprocessing can drastically improve the extent to which the input instance can be reduced, even when dealing with problems for which polynomial kernels have been ruled out under the existing theory of lower bounds. In particular, they showed that the CONNECTED VERTEX COVER problem admits an α -approximate polynomial kernelization for every $\alpha > 1$. Their result provided a promising starting point towards obtaining a refined understanding of the role played by connectivity constraints in relation to preprocessing for covering problems on graphs.

In this paper, we consider one of the most natural generalizations of CONNECTED VERTEX COVER, the CONNECTED FEEDBACK VERTEX SET (CFVS) problem defined as follows. In this problem, the input is a graph G and integer k (the parameter). The goal is to decide whether or not there is a set $S \subseteq V(G)$ of size at most k such that $G[S]$ is connected and $G - S$ is acyclic? The set S is called a connected feedback vertex set of G .

Misra et al. [23] were the first to study the CFVS problem from the point of view of parameterized complexity and obtained a single-exponential fixed-parameter algorithm, that is, an algorithm running in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$, where n is the number of vertices in the input. They also observed that a straightforward reduction from CONNECTED VERTEX COVER implies that CFVS is also unlikely to admit a polynomial kernelization under standard complexity theoretic hypotheses. This raises the natural question of the existence of approximate kernelizations for this problem and the tradeoffs between preprocessing speed, output size and loss in the quality of the preserved solution.

Our results

A formal definition of α -approximate kernels can be found in Section 2.

► **Theorem 1.** *For every fixed $0 < \varepsilon < 1$, CONNECTED FEEDBACK VERTEX SET has a $(1 + \varepsilon)$ -approximate kernelization of polynomial size.*

Note that the exponent in the size of our kernelization depends on ε .

The proof techniques we use in order to prove Theorem 1 also lead to a polynomial time approximation for this problem (Theorem 3) via the following *parameterized* approximation.

► **Theorem 2.** *There is an algorithm that given a graph G and $k \in \mathbb{N}$, runs in polynomial time and either correctly concludes that G has no connected feedback vertex set of size at most k or returns a connected feedback vertex set of G of size $k^{\mathcal{O}(1)}$.*

As a direct consequence of Theorem 2, we obtain the following result.

► **Theorem 3.** *There is a $0 < \delta < 1$ such that CONNECTED FVS can be approximated within a factor $\min\{\text{OPT}^{\mathcal{O}(1)}, n^{1-\delta}\}$ in polynomial time.*

Proof. By iteratively invoking Theorem 2 for $k = 1, \dots, n$, one can find the least k for which the algorithm *does not* return a negative answer and returns a connected feedback vertex set of the input graph of size at most k^c for some fixed $c > 0$. Since $\text{OPT} \geq k$, it follows that the returned solution has size at most OPT^c . Moreover, this algorithm always returns a solution whose size is bounded by n . The theorem follows from fact that the approximation ratio guaranteed by this algorithm is at most $\min\{\text{OPT}^{c-1}, \frac{n}{\text{OPT}}\} \leq n^{1-\frac{1}{c}}$. ◀

Our approximation result complements the classic result of Yannakakis [25] who showed that it is NP-hard to approximate within a factor- $\mathcal{O}(n^{1-\delta})$ (for any $\delta > 0$) the minimum number of vertices to delete from a graph such that the *resulting graph* is connected and has a property Π which is hereditary, non-trivial, “interesting” on connected graphs, and is determined by the blocks of the graph. In particular, this result holds if $\Pi = \text{acyclicity}$.

Related work on connected hitting set problems. Grigoriev and Sitters [16] studied the design of approximation algorithms for the CONNECTED FEEDBACK VERTEX SET problem on planar graphs and obtained a Polynomial Time Approximation Scheme (PTAS), building upon the result of Escoffier et al. [13] for CONNECTED VERTEX COVER. Eiben et al. [11] obtained an approximate kernelization for the CONNECTED \mathcal{H} -HITTING SET problem where \mathcal{H} is a fixed set of graphs and the solution is a minimum set of vertices which induces a connected subgraph and hits all copies of graphs in \mathcal{H} , in G . Recently, Eiben et al. [12] obtained approximate kernelizations for the CONNECTED DOMINATING SET problem on various sparse graph classes.

2 Preliminaries

A set $S \subseteq V(G)$ such that $G - S$ is a forest is called a *feedback vertex set* of G . For a path P , we denote by $V_{\text{int}}(P)$ the set of internal vertices of the path P . Similarly, we denote by $V_{\text{end}}(P)$ the set of endpoints of P . Two paths P_1 and P_2 are said to be *internally vertex disjoint* if $V_{\text{int}}(P_1) \cap V_{\text{int}}(P_2) = \emptyset$. For a graph G , we denote by $\mathcal{CC}(G)$ the set of connected components of G . For ease of presentation, we will abuse notation and interchangeably refer to $X \subseteq V(G)$ both as a vertex set and as a connected component of G if clear from the context.

► **Definition 4.** *Let G be a graph and $x, y \in V(G)$. Let \mathcal{P} be a set of internally vertex-disjoint x - y paths in G . Then, we call \mathcal{P} an x - y flow. The value of this flow is $|\mathcal{P}|$.*

Recall that Menger's theorem states that for distinct *non-adjacent* vertices x and y , the size of the smallest x - y separator is precisely the value of the maximum x - y flow in G .

Parameterized problems and (approximate) kernels. A parameterized problem Π is a subset of $\Gamma^* \times \mathbb{N}$ for some finite alphabet Γ . An instance of a parameterized problem consists of a pair (x, k) , where k is called the parameter. We assume that k is *given in unary* and hence $k \leq |x|$.

► **Definition 5 (Kernelization).** *Let $\Pi \subseteq \Gamma^* \times \mathbb{N}$ be a parameterized problem and g be a computable function. We say that Π admits a kernel of size g if there exists an algorithm referred to as a kernelization (or a kernel) that, given $(x, k) \in \Gamma^* \times \mathbb{N}$, outputs in time polynomial in $|x| + k$, a pair $(x', k') \in \Gamma^* \times \mathbb{N}$ such that (a) $(x, k) \in \Pi$ if and only if $(x', k') \in \Pi$, and (b) $\max\{|x'|, k'\} \leq g(k)$. If $g(k) = k^{\mathcal{O}(1)}$ then we say that Π admits a polynomial kernel.*

► **Definition 6 ([22]).** *A parameterized optimization (minimization or maximization) problem is a computable function $\Pi : \Sigma^* \times \mathbb{N} \times \Sigma^* \rightarrow \mathbb{R} \cup \{\pm\infty\}$.*

The *instances* of a parameterized optimization problem Π are pairs $(I, k) \in \Sigma^* \times \mathbb{N}$, and a *solution* to (I, k) is simply a string $s \in \Sigma^*$, such that $|s| \leq |I| + k$. The *value* of the solution s is $\Pi(I, k, s)$.

Since we only deal with a minimization problem in this work, we state some of the definitions only in terms of minimization problems when the definition for maximization problems is analogous. The parameterized optimization version of CONNECTED FEEDBACK VERTEX SET is a minimization problem with the optimization function $\text{CFVS} : \Sigma^* \times \mathbb{N} \times \Sigma^* \rightarrow \mathbb{R} \cup \{\infty\}$ defined as follows.

$$\text{CFVS}(G, k, S) = \begin{cases} \infty & \text{if } S \text{ is not a connected feedback vertex set of } G, \\ \min\{|S|, k + 1\} & \text{otherwise.} \end{cases}$$

► **Definition 7 ([22]).** *For a parameterized minimization problem Π , the optimum value of an instance $(I, k) \in \Sigma^* \times \mathbb{N}$ is $\text{OPT}_{\Pi}(I, k) = \min_{\substack{s \in \Sigma^* \\ |s| \leq |I| + k}} \Pi(I, k, s)$.*

Consequently, in the case of CONNECTED FEEDBACK VERTEX SET, we define

$$\text{OPT}(G, k) = \min_{S \subseteq V(G)} \text{CFVS}(G, k, S).$$

► **Remark 8** (Restricting our interest to solutions of size at most k). A reader encountering this particular definition of parameterized optimization problems for the first time might find the choice of $k + 1$ as a threshold a bit counter-intuitive because when one combines it with the natural notion of approximate solutions in the most intuitive way, the size of the solution would appear to always exceed $k + 1$, thus being normalized by this explicit threshold.

However, this definition is in fact equivalent (upto constant factors) to the more seemingly natural definition and in addition allows us to define the problem we are tackling independently of the (approximation factor of) algorithms for the problem. An additional point which we encourage the reader to keep in mind is the following. We consider k as a threshold; for solutions of size at most k we care about what their size is, while all solutions of size larger than k are *equally bad* in our eyes, and are consequently assigned value $k + 1$. We point the interested reader to Section 2.1, [22] and Section 3.2, [21] for an in-depth discussion of these definitions and their motivations.

We now recall the other relevant definitions from [22] regarding *approximate kernels*.

► **Definition 9** ([22]). Let $\alpha \geq 1$ be a real number and Π be a parameterized minimization problem. An α -approximate polynomial time preprocessing algorithm \mathcal{A} for Π is a pair of polynomial-time algorithms. The first one is called the reduction algorithm, and computes a map $\mathcal{R}_{\mathcal{A}} : \Sigma^* \times \mathbb{N} \rightarrow \Sigma^* \times \mathbb{N}$. Given as input an instance (I, k) of Π the reduction algorithm outputs another instance $(I', k') = \mathcal{R}_{\mathcal{A}}(I, k)$.

The second algorithm is called the solution lifting algorithm. This algorithm takes as input an instance $(I, k) \in \Sigma^* \times \mathbb{N}$ of Π , the output instance (I', k') of the reduction algorithm, and a solution s' to the instance (I', k') . The solution lifting algorithm works in time polynomial in $|I|, k, |I'|, k'$ and s' , and outputs a solution s to (I, k) such that $\frac{\Pi(I, k, s)}{\text{OPT}(I, k)} \leq \alpha \cdot \frac{\Pi(I', k', s')}{\text{OPT}(I', k')}$.

The size of a polynomial time preprocessing algorithm \mathcal{A} is a function $\text{size}_{\mathcal{A}} : \mathbb{N} \rightarrow \mathbb{N}$ defined as $\text{size}_{\mathcal{A}}(k) = \sup\{|I'| + k' : (I', k') = \mathcal{R}_{\mathcal{A}}(I, k), I \in \Sigma^*\}$.

► **Definition 10** ([22], α -approximate kernelization). An α -approximate kernelization (or α -approximate kernel) for a parameterized optimization problem Π , and real $\alpha \geq 1$, is an α -approximate polynomial time preprocessing algorithm \mathcal{A} for Π such that $\text{size}_{\mathcal{A}}$ is upper bounded by a computable function $g : \mathbb{N} \rightarrow \mathbb{N}$. We say that \mathcal{A} is an α -approximate polynomial kernelization if g is a polynomial function.

► **Definition 11** ([22]). A polynomial size approximate kernelization scheme (PSAKS) for a parameterized optimization problem Π is a family of α -approximate polynomial kernelization algorithms, with one such algorithm for every $\alpha > 1$.

Least Common Ancestor-Closure of sets in trees. For a rooted tree T and vertex set $M \subseteq V(T)$ the least common ancestor-closure (*LCA-closure*) $\text{LCA-closure}(M)$ is obtained by the following process. Initially, set $M' = M$. Then, as long as there are vertices x and y in M' whose least common ancestor w is not in M' , add w to M' . When the process terminates, output M' as the LCA-closure of M . The following folklore lemma summarizes the properties of LCA-closures which we will use in this paper.

► **Lemma 12.** Let T be a rooted tree, $M \subseteq V(T)$ and $M' = \text{LCA-closure}(M)$. Then $|M'| \leq 2|M|$ and for every connected component C of $T - M'$, $|N(C)| \leq 2$. Moreover, the number of connected components of $T - M'$ which have exactly 2 neighbors in M' is at most $|M'| - 1$.

► **Observation 2.1.** Let T be a rooted tree and T' a subtree of T with a unique neighbor in $V(T) \setminus V(T')$. If $M \subseteq V(T) \setminus V(T')$, then $\text{LCA-closure}(M)$ is disjoint from $V(T')$.

3 Overview of our Techniques

This section is devoted to an overview of the proof techniques we use to obtain our results. Fix $0 < \varepsilon < 1$ and let (G, k) be the input. Our initial objective is to identify a partition $(\mathcal{A}, \mathcal{B}, \mathcal{C})$ of $V(G)$, where $|\mathcal{B}| = k^{\mathcal{O}(1)}$, $G - \mathcal{B}$ is acyclic and there are no edges between \mathcal{A} and \mathcal{C} . In other words, \mathcal{B} separates \mathcal{A} and \mathcal{C} . Moreover, we will be able to prove that the vertices in \mathcal{C} only play the role of “connectors” and removing them from a connected feedback vertex set S of G may disconnect $G[S]$, but will still leave a subset of S which hits all cycles in G . On the other hand, the interaction of vertices in \mathcal{A} with the solution S could be much more complex. However, the number of connected components of $G[\mathcal{A}]$ will be shown to be $k^{\mathcal{O}(1)}$ and these can be shown to have a highly structured neighborhood in \mathcal{B} . For instance, we will be able to ensure (after a small modification to G) that the neighborhood of any connected component of $G[\mathcal{A}]$ can be partitioned into 2 sets T and J such that T is part of *every* feedback vertex set of G and $|J| = 2$. For every such component, the sets T and J can be efficiently computed from \mathcal{A} .

Once we have this partition in hand, we focus on each connected component of $G[\mathcal{A}]$ separately and from each component we identify a set of $k^{\mathcal{O}(f(1/\varepsilon))}$ vertices which, together with \mathcal{B} and a $k^{\mathcal{O}(f(1/\varepsilon))}$ sized subset of \mathcal{C} cover a $(1 + \varepsilon)$ -approximate solution. Finally, the remaining vertices are discarded by either deleting or contracting edges as appropriate. We note that the high level approach of trying to identify “hitters” and “connectors” among the vertices is quite natural and has been used in other work [22, 11, 12]. However, the structure is much more complex in our case due to the highly non-local nature of the forbidden structures (cycles) and the fact that there is no clear way of completely separating the two tasks of hitting cycles and connectivity. In fact, the main difficulty arises from the need to detect and control subsets of vertices which are critical with respect to both objectives simultaneously.

We now proceed to a slightly more detailed overview of the steps in our algorithm. Let δ be a positive constant such that $(1 + \delta)^3 \leq 1 + \varepsilon$. We also fix a constant $\rho = 2^{\mathcal{O}(1/\delta)^2}$ satisfying certain appropriate inequalities. As mentioned, our PSAKS for CONNECTED FEEDBACK VERTEX SET has two main parts: a structural decomposition and a reduction using marking rules.

3.1 Structural Decomposition

The first part of our PSAKS is the decomposition given by Lemma 13 below (also see Figure 1). To get the required partition $(\mathcal{A}, \mathcal{B}, \mathcal{C})$ we set $\mathcal{B} = H \cup X \cup Z$ where H, X, Z are as defined in the statement of Lemma 13. We then set \mathcal{C} to be the set of vertices in connected components of $G - (H \cup X \cup Z)$ which are adjacent to at most one vertex of Z and \mathcal{A} to be the rest of the vertices in $G - (H \cup X \cup Z)$.

► **Lemma 13.** *There is a polynomial-time algorithm that given a pair (G, k) , either correctly concludes that G has no connected feedback vertex set of size at most k or outputs pairwise vertex disjoint subsets $H, X, Z \subseteq V(G)$ satisfying the following properties.*

1. $|H| \leq (1 + \frac{\delta}{2})k$, $|X| = \mathcal{O}(k)$, $|Z| = \mathcal{O}(k^6)$.
2. $H \cup X$ is a feedback vertex set of G and if G has a connected feedback vertex set S of size at most k , then there is one of size at most $(1 + \frac{\delta}{2})|S|$ that contains H .
3. Every connected component of the graph $\widetilde{G}_Z = G - (H \cup X \cup Z)$ is adjacent to at most 2 vertices of Z and there are $\mathcal{O}(k^6)$ connected components in \widetilde{G}_Z which are adjacent to exactly 2 vertices of Z .

4. No vertex of X has a neighbor in a connected component of \widetilde{G}_Z which is adjacent to 2 vertices of Z and moreover, every vertex in any such component is adjacent to vertices in $\mathcal{O}(1/\delta)$ connected components of $G[H]$.
5. For every connected component D in the graph \widetilde{G}_Z with at most 1 neighbor in Z and for every minimal feedback vertex set S of $G - H$ of size at most $2k$, $|N(D) \setminus (H \cup S)| \leq 1$.

Proof. (Sketch for the construction of H, X, Z). We only sketch the construction of these sets here.

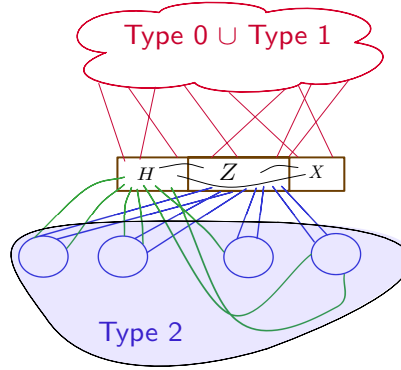
Step 1: This step is inspired by Fomin et al. [14]. However, since we need to handle connectivity constraints, we need to enhance the output of this step further with several problem specific features. Let $H = \{v_1, \dots, v_r\}$ be a maximal set such that for every $i \in [r]$, there is a v_i -flower of order $2k + 1$ ($2k + 1$ cycles which are pairwise disjoint except for intersecting at v_i) in $G - H_{i-1}$, where $H_i = \{v_1, \dots, v_i\}$ and $H_0 = \emptyset$. It is known that there is a polynomial time algorithm that, given G, v, ℓ (assuming v is not incident on a self-loop) either outputs a v -flower of order ℓ or a set $X \subseteq V(G) \setminus \{v\}$ of $\mathcal{O}(\ell)$ vertices intersecting every cycle containing v [24]. Hence H can be computed in polynomial time. Furthermore, it can be observed that every vertex in H is part of every feedback vertex set of size at most $2k$. Therefore, if $|H| > k$, then we may correctly conclude that G has no feedback vertex set of size at most k .

Let Q be a feedback vertex set of G computed using the 2-approximation algorithm of Bafna et al. [1]. If $|Q| > 2k$, then we may correctly conclude that G has no feedback vertex set of size at most k . Otherwise, $H \subseteq Q$ and we define $X = Q \setminus H$. Due to the maximality of H , it follows that for every $x \in X$, there is a set P_x which is disjoint from x , has size $\mathcal{O}(k)$ and intersects every cycle containing x in $G - H$.

For every pair $x, y \in X$ such that there is no x - y flow of value $2k + 3$ in the graph $G_{xy} = G - (H \cup (X \setminus \{x, y\}))$, we denote by Z_{xy} an arbitrarily chosen minimum x - y separator in the graph G'_{xy} , where $G'_{xy} = G_{xy}$ if $(x, y) \notin E(G)$ and $G'_{xy} = G_{xy} - (x, y)$ otherwise. By Menger's Theorem, for every such pair x, y , the size of the set Z_{xy} is at most $2k + 2$. We define $J = (\bigcup_{x \in X} P_x \cup \bigcup_{x, y \in X} Z_{xy}) \setminus (X \cup H)$, where for every pair $x, y \in X$ such that there is an x - y flow of value at least $2k + 3$ in the graph $G_{xy} = G - (H \cup (X \setminus \{x, y\}))$, Z_{xy} is defined to be \emptyset . We now define $Y = \text{LCA-closure}(J)$ with the LCA-closure taken in the graph $G - (H \cup X)$ where each tree is arbitrarily rooted. Since $|H| \leq k$ and $|X| \leq 2k$, it follows that $|J| = \mathcal{O}(k^3)$ and Lemma 12 implies that $|Y| = |\text{LCA-closure}(J)| = \mathcal{O}(k^3)$, every connected component of the graph $G - (H \cup X \cup Y)$ is adjacent to at most 2 vertices of Y and there are $\mathcal{O}(|Y|)$ of these components which are adjacent to exactly 2 vertices of Y .

Step 2: Since Step 1 guarantees that there are no cycles in $G - (H \cup Y \cup (X \setminus \{x\}))$ for any $x \in X$ we conclude that no vertex of X can have 2 neighbors in any connected component of $\widetilde{G}_Y = G - (H \cup X \cup Y)$. Let \mathcal{R} denote the set of all connected components of \widetilde{G}_Y which have no neighbors in Y , let \mathcal{Q} denote the set of all connected components of \widetilde{G}_Y which have exactly 1 neighbor in Y , and let \mathcal{W} denote the set of all connected components of \widetilde{G}_Y which have exactly 2 neighbors in Y .

For every $x \in X$ and $y \in Y$, we define the set \mathcal{J}_{xy} as the set of components of \widetilde{G}_Y which are adjacent to x and whose neighborhood in Y is exactly $\{y\}$. We say that the set $\mathcal{J}_{x,y}$ is *rich* if $|\mathcal{J}_{xy}| \geq 2k + 3$ and *poor* otherwise. We call a connected component in \mathcal{Q} *poor* if it appears in *at least one* poor set and *rich* otherwise. Let $\mathcal{Q}_{\text{poor}}$ denote the set of poor components in \mathcal{Q} . By definition, the size of the set $\mathcal{Q}_{\text{poor}}$ is bounded by $|X| \cdot |Y| \cdot (2k + 2) = \mathcal{O}(k^5)$. Let P_1 denote the neighborhood of X in the set of components



■ **Figure 1** An illustration of the decomposition guaranteed by Lemma 13. The (blue) circles below the set $(H \cup Z \cup X)$ represent components of $G - (H \cup Z \cup X)$ with 2 neighbors in Z . Note that there are no edges between X and such a component.

in $\mathcal{Q}_{\text{poor}}$. Since every vertex of X has at most 1 neighbor in each of these components, the size of P_1 is at most $|X| \cdot |\mathcal{Q}_{\text{poor}}| = \mathcal{O}(k^6)$. Let P_2 denote the neighborhood of X in the set of components of G_Y with exactly 2 neighbors in Y . Since there are only $\mathcal{O}(k^3)$ such components (from Step 1), the size of P_2 is $\mathcal{O}(k^4)$.

Finally, we define $Z = \text{LCA-closure}(Y \cup P_1 \cup P_2)$ where the LCA-closure is taken in the graph $G - (H \cup X)$ with an arbitrary rooting of each tree. From Lemma 12, it follows that $|Z| = \mathcal{O}(k^6)$, every connected component of the graph $\widetilde{G}_Z = G - (H \cup X \cup Z)$ is adjacent to at most 2 vertices of Z and there are $\mathcal{O}(k^6)$ connected components in the graph \widetilde{G}_Z which are adjacent to exactly 2 vertices of Z . Moreover, we will be able to argue using flow arguments and the definition of rich/poor components that (i) no vertex of X has a neighbor in a connected component of \widetilde{G}_Z which is adjacent to 2 vertices of Z and (ii) for every connected component D in the graph \widetilde{G}_Z with at most 1 neighbor in Z and for any minimal feedback vertex set S of $G - H$ of size at most $2k$, S contains all but at most one vertex of $N(D) \setminus H$.

Step 3: We will finally augment the set H by adding some more vertices. Specifically, we will grow the set H by “buying a cheap set of vertices” as follows. As long as there is a vertex v which is adjacent to at least $\lceil \frac{2}{\delta} \rceil + 1$ connected components of $G[H]$ and contained in a connected component of \widetilde{G}_Z adjacent to 2 vertices of Z , we set $H := H \cup \{v\}$. When this process terminates, it must be the case that every vertex which is in a connected component of \widetilde{G}_Z adjacent to 2 vertices of Z , has at most $\lceil \frac{2}{\delta} \rceil$ neighboring components of $G[H]$. A simple counting argument based on the fact that we repeatedly decrease the number of connected components in $G[H]$ shows that the blow-up in the size of H is at most a $\delta/2$ fraction of the original value of $|H|$. ◀

We call a component D of \widetilde{G}_Z a **Type 0** component if $|N(D) \cap Z| = 0$, a **Type 1** component if $|N(D) \cap Z| = 1$ and a **Type 2** component if $|N(D) \cap Z| = 2$. Lemma 13 (5) implies that the Type 0 and Type 1 components (which comprise the set \mathcal{C}) only play the role of connectors and Lemma 13 (3) guarantees that the number of Type 2 components (which comprise the set \mathcal{A}) is $\mathcal{O}(k^6)$.

3.2 Marking Rules and Reduction Strategy via Steiner Trees

Once the partition $(\mathcal{A}, \mathcal{B}, \mathcal{C})$ is computed, the second part of the PSAKS relies on an extension of the following result of Du et al. [10] to a special case of the Group Steiner Tree problem where at most one group can have size greater than 1.

► **Proposition 14** ([10]). *For every $p \geq 1$, graph G , $R \subseteq V(G)$, cost function $w : E(G) \rightarrow \mathbb{N} \cup \{0\}$ and R -Steiner tree T , there is a p -restricted R -Steiner tree in G of cost at most $(1 + \frac{1}{\lfloor \log_2 p \rfloor}) \cdot w(T)$.*

In the above proposition, an R -Steiner tree is a subtree of G containing R and a p -restricted R -Steiner tree is a connected subgraph of G containing R and whose edge set can be written as the union of the edge sets of Steiner trees for some subsets of R of size at most p where these subsets appear as the leaves of the respective Steiner trees. In a similar spirit to Proposition 14, we show that for every $p \geq 1$, graph G , $R \subseteq V(G)$, cost function $w : E(G) \rightarrow \mathbb{N} \cup \{0\}$ and R -Steiner tree T intersecting a set \mathcal{Z} of arbitrary size disjoint from R , there is a $2p + 1$ -restricted R -Steiner tree in G of cost at most $(1 + \frac{1}{\lfloor \log_2 p \rfloor}) \cdot w(T)$ which also intersects \mathcal{Z} . We believe that this extension and close variants thereof are likely to have future applications in dealing with connectivity constraints. We design a set of marking rules that achieve the following.

► **Lemma 15.** *There is an algorithm that, given G , k and the partition $(\mathcal{A}, \mathcal{B}, \mathcal{C})$, runs in time $k^{\mathcal{O}(\rho)} n^{\mathcal{O}(1)}$ and marks a set $\mathcal{A}' \subseteq \mathcal{A}$ of $k^{\mathcal{O}(\rho)}$ vertices such that for any S which is a connected feedback vertex set of G of size at most k , there is a connected feedback vertex set of G of size at most $(1 + \delta)^2 |S|$ whose intersection with \mathcal{A} is contained in \mathcal{A}' .*

The main technical difficulty in the above lemma lies in identifying and marking vertices such that for any subset of $S \cap \mathcal{A}$ which may be performing the dual job of “hit” and “connect”, the set \mathcal{A}' of marked vertices contains a subset which would do the same job with only a small increase in size. Moreover, since we deal with each connected component of $G[\mathcal{A}]$ separately, we cannot simply use Proposition 14 or its extension we propose. This is where the upper bound on the degree of vertices of \mathcal{A} into the set H will be crucial (Lemma 13 (4)).

Using Lemma 15, we will show that there is a way to reduce the graph $G[\mathcal{A}]$ by deleting or contracting all but $k^{\mathcal{O}(\rho)}$ of the rest of the edges so that the only unbounded set following this step is the set \mathcal{C} . As we know that the vertices in \mathcal{C} only perform the job of “connectors” and are not necessary to hit cycles in G , we will mark the optimal Steiner tree (if it is small enough) in G for every choice of a sufficiently small subset of $\mathcal{A}' \cup \mathcal{B}$ as the set of terminals and use Proposition 14 to prove the following.

► **Lemma 16.** *There is an algorithm that, given G , k and the partition $(\mathcal{A}, \mathcal{B}, \mathcal{C})$, runs in time $k^{\mathcal{O}(\rho^2)} n^{\mathcal{O}(1)}$ and marks a set $\mathcal{C}' \subseteq \mathcal{C}$ of $k^{\mathcal{O}(\rho^2)}$ vertices such that for any S which is a connected feedback vertex set of G of size at most k , there is a connected feedback vertex set of G of size at most $(1 + \delta)^3 |S|$ whose intersection with \mathcal{C} is contained in \mathcal{C}' .*

Finally, we show that we can delete the unmarked vertices in \mathcal{C} after marking an additional $k^{\mathcal{O}(1)}$ new vertices and edges simply to remember the cycles passing through \mathcal{C} . Since we chose δ such that $(1 + \delta)^3 \leq 1 + \varepsilon$, we will have obtained the required graph. Specifically, we prove the following lemma.

► **Lemma 17.** *There is an algorithm that given G , k and the outputs of Lemma 13, Lemma 15 and Lemma 16 runs in time $k^{\mathcal{O}(\rho^2)} n^{\mathcal{O}(1)}$ and either correctly concludes that G has no connected feedback vertex set of size at most k or returns a graph G' such that:*

1. $|V(G')| = k^{\mathcal{O}(\rho^2)}$.
2. Every minimal connected feedback vertex set of G' of size at most $(1 + \varepsilon)k$ is contained in $V(G') \cap V(G)$ and is also a connected feedback vertex set of G .
3. For every S which is a minimal connected feedback vertex set of G of size at most k , G' has a connected feedback vertex set of size at most $(1 + \varepsilon)|S|$.

We summarize the steps of our algorithm below.

1. (Lemma 13) In polynomial time, identify a partition $(\mathcal{A}, \mathcal{B}, \mathcal{C})$ of $V(G)$ such that:
 - $|\mathcal{B}| = k^{\mathcal{O}(1)}$, $G - \mathcal{B}$ is acyclic, and \mathcal{B} separates \mathcal{A} and \mathcal{C} .
 - For every connected feedback vertex set S of G , $G - (S \setminus \mathcal{C})$ is acyclic.
 - Every connected component of $G[\mathcal{A}]$ has exactly 2 neighbors in \mathcal{B} and there are $k^{\mathcal{O}(1)}$ connected components in $G[\mathcal{A}]$.
2. (Lemma 15 + Lemma 16) In time $k^{f(1/\varepsilon)}n^{\mathcal{O}(1)}$, mark sets $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{C}' \subseteq \mathcal{C}$ of $k^{\mathcal{O}(f(1/\varepsilon))}$ vertices such that for any S which is a connected feedback vertex set of G of size at most k , there is a connected feedback vertex set of G of size at most $(1 + \varepsilon)|S|$ whose intersection with \mathcal{A} (\mathcal{C}) is contained in \mathcal{A}' (\mathcal{C}').
3. (Lemma 17) In time $k^{f(1/\varepsilon)}n^{\mathcal{O}(1)}$, compute a graph G' with $k^{\mathcal{O}(f(1/\varepsilon))}$ vertices where:
 - Every minimal connected feedback vertex set of G' of size $\leq (1 + \varepsilon)k$ is contained in $V(G') \cap V(G)$ and is also a connected feedback vertex set of G .
 - For every S which is a minimal connected feedback vertex set of G of size at most k , G' has a connected feedback vertex set of size at most $(1 + \varepsilon)|S|$.

3.3 The PSAKS and Factor-OPT ^{$\mathcal{O}(1)$} Approximation

We are now ready to prove Theorem 1 by translating Lemma 17 into a PSAKS for CONNECTED FEEDBACK VERTEX SET under the framework from [22]. Recall the definition of the parameterized optimization version of CONNECTED FEEDBACK VERTEX SET has the optimization function $\text{CFVS} : \Sigma^* \times \mathbb{N} \times \Sigma^* \rightarrow \mathbb{R} \cup \{\infty\}$ defined as follows.

$$\text{CFVS}(G, k, S) = \begin{cases} \infty & \text{if } S \text{ is not a connected feedback vertex set of } G \\ \min\{|S|, k + 1\} & \text{otherwise} \end{cases}$$

We define $\text{OPT}(G, k) = \min_{S \subseteq V(G)} \text{CFVS}(G, k, S)$. We point out that ∞ can be replaced in our setting with a sufficiently large value depending on $|V(G)|$ while serving the same purpose and avoiding potentially undefined operations.

► **Theorem 1.** *For every fixed $0 < \varepsilon < 1$, CONNECTED FEEDBACK VERTEX SET has a $(1 + \varepsilon)$ -approximate kernelization of polynomial size.*

Proof. We begin by describing the reduction algorithm. If G is a forest, then return $(G', 0)$ where G' is the empty graph. Henceforth we ignore this corner case and assume that G contains a cycle. Given G, k and ε , we execute the algorithm of Lemma 17. If this algorithm concludes that G does not contain a connected feedback vertex set of size at most k , then we return the instance (G', k) where G' is a connected graph with $k + 1$ vertex disjoint cycles and having $\mathcal{O}(k)$ vertices. Clearly such a graph exists. For instance, take the disjoint union of $k + 1$ triangles and connect them in the form of a path by selecting exactly one vertex from each triangle. Otherwise, suppose that this algorithm returns a graph G' . Then the reduction algorithm returns the instance (G', k') where $k' = (1 + \varepsilon)k$. At this point, we may assume that G is connected and so $V(G)$ is a connected feedback vertex set of G . If G' has multiple connected components containing cycles, then it cannot have a connected feedback

vertex set of *any* size, implying (by Lemma 17 (3)) that G cannot have a connected feedback vertex set of size at most k . Therefore we fall back into the previous case and so we may assume going forward that G' is also connected (if there are multiple components but only one contains cycles, then discard the rest) and so $V(G')$ is a connected feedback vertex set of G' . The polynomial bound on the size of the output instance and the time required to obtain G' follow from Lemma 17.

We now describe the solution lifting algorithm as follows. Let S' be the given solution for (G', k') . If S' is not a connected feedback vertex set of G' , then the algorithm outputs \emptyset as the solution for (G, k) . If S' is a connected feedback vertex set of G' and $|S'| > k'$, then the algorithm outputs $V(G)$ as the solution for (G, k) . Finally, if S' is a connected feedback vertex set of G' and $|S'| \leq k'$, then the algorithm returns S' as the solution for (G, k) . We denote by S the output of the solution lifting algorithm. Clearly, the solution lifting algorithm runs in polynomial time.

We now prove that this reduction algorithm and the solution lifting algorithm together constitute a $(1 + \varepsilon)$ -approximate kernelization for CONNECTED FEEDBACK VERTEX SET. We show that if $\text{CFVS}(G', k', S') \leq c \cdot \text{OPT}(G', k')$, then $\text{CFVS}(G, k, S) \leq (1 + \varepsilon)c \cdot \text{OPT}(G, k)$. In the case where we concluded that G has no connected feedback vertex set of size at most k and returned (G', k) , notice that $\text{CFVS}(G', k, S') = \text{CFVS}(G, k, S)$ and $\text{OPT}(G, k) = \text{OPT}(G', k) = k + 1$. Hence, we are left with the following cases which arise when the invocation to Lemma 17 returned a graph G' . We will use the fact that $\text{OPT}(G', k') \leq (1 + \varepsilon)\text{OPT}(G, k)$.

Case 1: S' is a connected feedback vertex set of G' and $|S'| > k'$. In this case, $S = V(G)$.

We now consider two subcases: G has a connected feedback vertex set of size at most k or it does not. If G has no connected feedback vertex set of size at most k , it follows that $\text{OPT}(G, k) = \text{CFVS}(G, k, V(G)) = k + 1$. This is because G is connected and so $V(G)$ is a connected feedback vertex set. Therefore,

$$\frac{\text{CFVS}(G, k, V(G))}{\text{OPT}(G, k)} = 1 \leq (1 + \varepsilon) \cdot \frac{\text{CFVS}(G', k', S')}{\text{OPT}(G', k')}. \quad (1)$$

Hence, we may assume that G has a connected feedback vertex set of size at most k . That is, $\text{OPT}(G, k) \leq k$. But this implies the following.

$$\frac{\text{CFVS}(G, k, V(G))}{\text{OPT}(G, k)} = \frac{k + 1}{\text{OPT}(G, k)} \leq (1 + \varepsilon) \cdot \frac{k' + 1}{\text{OPT}(G', k')} = (1 + \varepsilon) \frac{\text{CFVS}(G', k', S')}{\text{OPT}(G', k')}. \quad (2)$$

Case 2: S' is a connected feedback vertex set of G' and $|S'| \leq k'$. In this case, $\text{OPT}(G', k') \leq k'$. We consider two subcases: $|S'| \leq k$ or $k + 1 \leq |S'| \leq k'$. In the former subcase we have the following.

$$\frac{\text{CFVS}(G, k, S)}{\text{OPT}(G, k)} = \frac{\text{CFVS}(G, k, S')}{\text{OPT}(G, k)} = \frac{\text{CFVS}(G', k', S')}{\text{OPT}(G, k)} \leq (1 + \varepsilon) \frac{\text{CFVS}(G', k', S')}{\text{OPT}(G', k')}. \quad (3)$$

And if $k + 1 \leq |S'| \leq k'$, then we have the following.

$$\begin{aligned} \frac{\text{CFVS}(G, k, S)}{\text{OPT}(G, k)} &= \frac{\text{CFVS}(G, k, S')}{\text{OPT}(G, k)} = \frac{k + 1}{\text{OPT}(G, k)} \leq (1 + \varepsilon) \frac{k + 1}{\text{OPT}(G', k')} \\ &\leq (1 + \varepsilon) \frac{\text{CFVS}(G', k', S')}{\text{OPT}(G', k')}. \end{aligned} \quad (4)$$

Case 3: S' is not a connected feedback vertex set of G' . Then, the set returned by the solution lifting algorithm, \emptyset , is also not a connected feedback vertex set of G and so, $\text{CFVS}(G', k', S') = \text{CFVS}(G, k, \emptyset)$. As a result, the required inequality follows and this completes the proof of the theorem. ◀

Lemma 17 also implies the following $\text{OPT}^{\mathcal{O}(1)}$ -approximation for CONNECTED FEEDBACK VERTEX SET.

► **Lemma 18.** *There is a polynomial time algorithm that given G, k , either correctly concludes that G has no connected feedback vertex set of size at most k or returns a connected feedback vertex set of G of size $k^{\mathcal{O}(1)}$.*

Proof. Given G, k , we set ε to be an arbitrary constant between 0 and 1, say $\frac{1}{2}$. We then invoke Lemma 17 to either correctly conclude that G has no connected feedback vertex set of size at most k or compute the graph G' guaranteed by the lemma. In the former case we return the same. Otherwise, we check for each connected component of $G[V(G') \cap V(G)]$ whether it is a connected feedback vertex set of G . If such a component exists, then we return the vertex set of this component and otherwise we conclude that G has no connected feedback vertex set of size at most k . The correctness follows from the fact that if G contains at least one connected feedback vertex set of size at most k then at least one connected component of $G[V(G') \cap V(G)]$ is guaranteed to contain a connected feedback vertex set of G according to Lemma 17 and such a connected component by itself must also be a connected feedback vertex set of G . This completes the proof of the lemma. ◀

4 Conclusions and Open Problems

Our result on approximate kernelization for CONNECTED FEEDBACK VERTEX SET provides another useful data point in improving our understanding of the extent to which (approximate) preprocessing can be performed in the presence of connectivity constraints. Moreover, we believe that our techniques could have further applications in the design of approximate kernels for covering problems with connectivity constraints. Finally, this line of investigation offers several interesting opportunities for further research.

For instance, is there a *space efficient* PSAKS for CONNECTED FEEDBACK VERTEX SET? In a space efficient PSAKS, we require the size of the output to be bounded by $f(\frac{1}{\varepsilon}) \cdot k^c$, where f is a computable function and c is a constant independent of the error parameter ε . Essentially, a PSAKS is an apt analogue of a PTAS in the approximate kernelization world and an Efficient PSAKS is a natural analogue of an Efficient PTAS in this setting. We end by pointing out that the existence of a space efficient PSAKS is open even in the case of the CONNECTED VERTEX COVER problem.

References

- 1 Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-Approximation Algorithm for the Undirected Feedback Vertex Set Problem. *SIAM J. Discrete Math.*, 12(3):289–297, 1999. doi:10.1137/S0895480196305124.
- 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.
- 3 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Cross-Composition: A New Technique for Kernelization Lower Bounds. In *28th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 165–176, 2011.

- 4 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 5 Holger Dell and Dániel Marx. Kernelization of packing problems. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 68–81, 2012.
- 6 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 251–260, 2010.
- 7 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization Lower Bounds Through Colors and IDs. *ACM Transactions on Algorithms*, 11(2):13:1–13:20, 2014.
- 8 Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.
- 9 Andrew Drucker. New Limits to Classical and Quantum Instance Compression. *SIAM J. Comput.*, 44(5):1443–1479, 2015.
- 10 Ding-Zhu Du, Yanjun Zhang, and Qing Feng. On Better Heuristic for Euclidean Steiner Minimum Trees (Extended Abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 431–439, 1991. doi:10.1109/SFCS.1991.185402.
- 11 Eduard Eiben, Danny Hermelin, and M. S. Ramanujan. Lossy Kernels for Hitting Subgraphs. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, pages 67:1–67:14, 2017. doi:10.4230/LIPIcs.MFCS.2017.67.
- 12 Eduard Eiben, Mithilesh Kumar, Amer E. Mouawad, Fahad Panolan, and Sebastian Siebertz. Lossy Kernels for Connected Dominating Set on Sparse Graphs. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 29:1–29:15, 2018. doi:10.4230/LIPIcs.STACS.2018.29.
- 13 Bruno Escoffier, Laurent Gourvès, and Jérôme Monnot. Complexity and approximation results for the connected vertex cover problem in graphs and hypergraphs. *J. Discrete Algorithms*, 8(1):36–49, 2010. doi:10.1016/j.jda.2009.01.005.
- 14 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar \mathcal{F} -Deletion: Approximation, Kernelization and Optimal FPT Algorithms. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 470–479, 2012. doi:10.1109/FOCS.2012.62.
- 15 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.
- 16 Alexander Grigoriev and René Sitters. Connected Feedback Vertex Set in Planar Graphs. In *Graph-Theoretic Concepts in Computer Science, 35th International Workshop, WG 2009, Montpellier, France, June 24-26, 2009. Revised Papers*, pages 143–153, 2009. doi:10.1007/978-3-642-11409-0_13.
- 17 Danny Hermelin, Stefan Kratsch, Karolina Soltys, Magnus Wahlström, and Xi Wu. A Completeness Theory for Polynomial (Turing) Kernelization. *Algorithmica*, 71(3):702–730, 2015.
- 18 Danny Hermelin and Xi Wu. Weak compositions and their applications to polynomial lower bounds for kernelization. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 104–113, 2012.
- 19 Stefan Kratsch. Recent developments in kernelization: A survey. *Bulletin of the EATCS*, 113, 2014.
- 20 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Kernelization–preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond*, pages 129–161. Springer, 2012.

- 21 Daniel Lokshantov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy Kernelization. *CoRR*, abs/1604.04111, 2016. [arXiv:1604.04111](https://arxiv.org/abs/1604.04111).
- 22 Daniel Lokshantov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 224–237, 2017. [doi:10.1145/3055399.3055456](https://doi.org/10.1145/3055399.3055456).
- 23 Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. FPT algorithms for connected feedback vertex set. *J. Comb. Optim.*, 24(2):131–146, 2012. [doi:10.1007/s10878-011-9394-2](https://doi.org/10.1007/s10878-011-9394-2).
- 24 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2):32:1–32:8, 2010. [doi:10.1145/1721837.1721848](https://doi.org/10.1145/1721837.1721848).
- 25 Mihalis Yannakakis. The Effect of a Connectivity Requirement on the Complexity of Maximum Subgraph Problems. *J. ACM*, 26(4):618–630, 1979. [doi:10.1145/322154.322157](https://doi.org/10.1145/322154.322157).

Multicommodity Multicast, Wireless and Fast

R. Ravi

Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA
<https://www.contrib.andrew.cmu.edu/~ravi/>
ravi@andrew.cmu.edu

Oleksandr Rudenko

Carnegie Mellon University, Pittsburgh, PA, USA
orudenko@andrew.cmu.edu

Abstract

We study rumor spreading in graphs, specifically multicommodity multicast problem under the wireless model: given source-destination pairs in the graph, one needs to find the fastest schedule to transfer information from each source to the corresponding destination. Under the wireless model, nodes can transmit to any subset of their neighbors in synchronous time steps, as long as they either transmit or receive from at most one transmitter during the same time step. We improve approximation ratio for this problem from $\tilde{O}(n^{\frac{2}{3}})$ to $\tilde{O}(n^{\frac{1}{2}+\epsilon})$ on n -node graphs. We also design an algorithm that satisfies p given demand pairs in $O(\text{OPT} + p)$ steps, where OPT is the length of an optimal schedule, by reducing it to the well-studied packet routing problem. In the case where underlying graph is an n -node tree, we improve the previously best-known approximation ratio of $O(\frac{\log n}{\log \log n})$ to 3. One consequence of our proof is a simple constructive rule for optimal broadcasting in a tree under a widely studied telephone model.

2012 ACM Subject Classification Networks → Routing protocols; Networks

Keywords and phrases Rumor, scheduling, broadcast, multicommodity, multicast, optimization algorithms, approximation, packet routing

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.78

Funding This material is based upon research supported in part by the U. S. Office of Naval Research under award number N00014-18-1-2099, and the U. S. National Science Foundation under award number CCF-1527032.

1 Introduction

1.1 Motivation and Formulation

Rumor spreading problems have been popular for decades motivated by applications ranging from increasing throughput in synchronizing networks [3], keeping object copies in distributed databases synchronized [11], to recreational mathematics [4]. Common objectives in rumor spreading involve the total number of messages, the total number of transmissions (especially when message sizes are bounded in transmissions) and the completion time. In this paper, we study the minimum completion time objective for the multicast version of the problem where a set of source terminals wish to send their messages to their respective subsets of sinks.

The requirements for rumor spreading problem range from sending a single message from a single source to all nodes (broadcast), a subset of nodes (multicast) or a more generalized multicommodity version of multicast that we study. The rules for message transmission that have been widely studied are synchronous and range from the telephone model [5], the radio model [1] and the recently introduced wireless [6] model that shares the features of the first two. We mainly study the wireless model in this paper.



© R. Ravi and Oleksandr Rudenko;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 78; pp. 78:1–78:20



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Multicommodity Multicast Problem. Given a communication graph G and demand pairs of vertices $(s_1, t_1), \dots, (s_p, t_p)$, where each source s_i has a unique piece of information (message or packet), the problem is to find a schedule that transfers the message from every source s_i to its corresponding destination t_i in a minimum number of steps. The rules under which the information is allowed to be sent in each step depend on the model.

Wireless Model. During each step, every node v in G is allowed to pick some of the vertices adjacent to v and send all information v has accumulated to those nodes at once (so there is no bound on message size). Nodes cannot receive from more than one source, nor can they send and receive during the same time step.

1.2 Related Work

The type of models that have been studied could be differentiated by the demand requirements and the rules under which information is spread. We highlight three different models relevant to our study, which roughly correspond to spanning, Steiner and generalized Steiner connectivity requirements studied in network design.

1. In the **Broadcast** problem, there is a single source (root) that must send its information to *all* nodes in the system.
2. In the more general **Multicast** problem, the root only needs its information to reach a *subset* of nodes in the system.
3. In the even more general **Multicommodity Multicast** problem, we are given source-destination pairs, and *every* source must transmit its message to the corresponding destination node.

We describe various models that define the rules of synchronous information transmission in rounds.

1. In the **Telephone model**, during one time step, every node with information is allowed to send it to *only one* of the adjacent nodes (neighbours) in the graph.
2. In the **Radio model**, a set of transmitting nodes send out information and only the other nodes that have unique transmitting neighbors can receive the information from that neighbor¹.
3. In the **Wireless model**, during one time step, every node with information is allowed to send it to *any* subset of neighbours.

In all models, nodes cannot receive from more than one source, nor can they send and receive during the same time step. The key difference between the radio and wireless models is that in the wireless model, receivers have tunable apparatus that allows them to listen in one exactly one of the neighbors that may be transmitting, whereas in the radio model, a node with two neighbors transmitting cannot receive any information due to interference. The most widely studied versions do not restrict the number of messages sent between a pair of nodes in each step, but capacitated versions are possible.

1.2.1 Telephone model

For the *broadcast/multicast* problems under the telephone model Ravi [10] showed a poly-logarithmic approximation in general graphs, by relating it to spanning/Steiner trees that simultaneously have small max degree and diameter (the so-called *poise* of the graph).

¹ We do not discuss this model further in this paper.

For this *broadcast/multicast* problem under the telephone model Elkin and Kortsarz [2] gave the best known $O(\frac{\log k}{\log \log k})$ -approximation factor, where k is the number of terminals the information should be delivered to.

► **Theorem 1** (Multicast approximation [2]). *Given graph G , root r and k target sinks, there is a poly-time algorithm TELEPHONEMULTICAST that finds a schedule that sends information from the root to all sinks in the telephone model and has approximation ratio $O(\frac{\log k}{\log \log k})$.*

The very general *multicommodity multicast* under telephone model was studied as well, where the best approximating ratio $\tilde{O}(2\sqrt{\log k})$ was achieved by Nikzad and Ravi [9]. Note that this ratio is super-polylogarithmic but sub polynomial in k , the number of terminals. No better approximations are known and part of the difficulty stems from the inability to relate this objective directly to graph parameters since the solution can in general be disconnected. This problem was also studied for special classes of graphs. First, it is known that if G is a tree, then a polynomial time algorithm to find optimal schedule for telephone broadcast can be derived through dynamic programming. Recently Iglesias, Rajaraman, Ravi and Sundaram [7] have shown a $O(\frac{\log^3 k \log n}{\log \log n})$ -approximation for multicommodity multicast for planar graphs.

1.2.2 Wireless model

Given that there are no restrictions on the number of neighbors receiving a transmission, for both the *broadcast and multicast* problems, optimal schedules under wireless model can be found in polynomial time using a simple *Breadth-First Search* (BFS) algorithm.

For the *multicommodity multicast* problem the best known approximate ratio was given by Iglesias, Rajaraman, Ravi and Sundaram [6], who also introduced the wireless model. They also showed an $O(\frac{\log n}{\log \log n})$ -approximation when the underlying graph is an n -node tree, as well as $\tilde{O}(n^{\frac{2}{3}})$ -approximation for general n -node graphs. There were two main ideas used in their paper: (i) If there are many sources that want to send to the same sink, we can satisfy them quickly using approximation algorithm for multicast under the telephone model; (ii) If there are many short disjoint path from sources to their respective destinations, we can satisfy them in parallel, otherwise we can reduce the number of sources considerably.

1.3 Packet routing problem

Another widely studied problem related to our work is so-called *store-and-forward packet routing problem*, where given packets at different source nodes, one needs to find a synchronous schedule of transporting them to given destinations, with the only restriction that no two packets can traverse the same edge at the same time step.

The main difference between store-and-forward packet routing and multicommodity multicast under wireless model is that the capacity restriction is put on the edge, not on the node. This means that a node can do the following: (i) receive several packets at the same time; (ii) send several packets as long as they use different edges; (iii) send and receive at the same time. What is not allowed in this model is to accumulate several packets and send them along one edge altogether, in contrast with the wireless model which can do that.

The analysis of packet routing problem involves a trade-off between the “dilation” parameter d (maximum path length) and the “congestion” criterion c (maximum number of paths using any edge). Note, that both c and d are lower bounds on the length of the optimal schedule for packet routing. Srinivasan and Teo [12] showed a polynomial time algorithm to find a schedule of length $O(c + d)$, which achieves a constant approximation. We reduce wireless multicommodity multicast problem to a packet routing problem to derive one of our results.

When the paths for the packets are given, Leighton, Maggs and Rao [8] initiated a long line of work that showed the existence and efficient construction of schedules of length $O(c + d)$, which was used extensively in the work cited above on finding such near optimal routes when the paths are not specified in advance.

1.4 Our Contributions

In this paper we focus on *multicommodity multicast* problem under the *wireless model*.

- We show that for any given constant $\varepsilon > 0$, there is a polynomial time $\tilde{O}(n^{\frac{1}{2}+\varepsilon})$ -approximation algorithm that, given an arbitrary graph G and pairs of source-destination nodes (s_i, t_i) , finds a schedule to transfer information from source s_i to destination t_i for all i (Theorem 2). This algorithm generalizes the ideas of Iglesias et al. [6] and applies them recursively.
- We design a polynomial time algorithm that, given an arbitrary graph G and p pairs of source-destination nodes (s_i, t_i) , finds a schedule of length $O(\text{OPT} + p)$ to transfer information from source s_i to destination t_i for all i , where OPT is the length of the optimal one (Theorem 10). To prove this result, we reduce our problem to an instance of packet routing in an appropriately defined auxiliary digraph.
- We show that there is a polynomial time 3-approximation algorithm that, given a tree T and pairs of source-destination nodes (s_i, t_i) , finds a schedule to transfer information from source s_i to destination t_i for all i (Theorem 16). This result decomposes the schedule into sending messages up the tree followed by a phase that sends it down the tree.
- We give a **simple** optimal schedule for the broadcast in the tree under telephone model (Theorem 23). This is not a new result but a conceptual improvement over the previous strategies that all rely on dynamic programming by providing a simple explicit rule for choosing the transmitting pairs.

The widely studied telephone and relatively new wireless models are related to each other in the following way. Even though spreading information from one source to several sinks under wireless model can be done efficiently using BFS, collecting information from several sources into one sink is equivalent under both models. This follows from the fact that while collecting to one sink every piece of information travels on a straight path. This observation implies that the *gossip problem*, where every node in the graph has to sent its message to all other nodes, is equivalent under both models, up to a constant factor. More precisely, the optimal schedule for the gossip problem under the wireless model is no longer than twice the optimal broadcast schedule under the telephone model, which can be used to sweep all information to a fixed root and spread it back in a BFS tree to all nodes. In the other direction, the optimal schedule for the gossip problem under the wireless model is at least as long as the optimal broadcast schedule under the telephone model since both models are required to collect information from all the nodes to a root under the same constraints.

The multicommodity multicast problem is a generalization of both multicast and gossip problems, so combines the difficulty of both. Finding approximation algorithms for this problem under wireless model includes difficulties for multicommodity multicast under telephone model. As mentioned earlier, the best-known approximation ratio of $\tilde{O}(2\sqrt{\log n})$ [9] for multicommodity multicast under telephone model is sub polynomial in n , whereas the best-known ratio for this problem under the wireless model is $\tilde{O}(n^{\frac{2}{3}})$ [6]. Thus, the very general multicommodity multicast problem under the wireless model appears to be quite hard to approximate. Our work significantly improve this ratio down to roughly $O(\sqrt{n})$.

2 General graphs

In this section, we study general multicommodity-multicast problem in arbitrary graphs. The best known approximation ratio so far is $\tilde{O}(n^{\frac{2}{3}})$ [6]. We improve this bound to roughly $\tilde{O}(\sqrt{n})$.

► **Theorem 2.** *Algorithm 3 is a polynomial algorithm that, given any $\varepsilon > 0$ and arbitrary graph with demand pairs, can find a multicommodity-multicast schedule in the wireless model with approximation ratio $O(n^{\frac{1}{2}+\varepsilon})$.*

In the sequel, we assume that we are given the length L of the optimal rumor spreading schedule. Note that there is always a wireless multicast schedule of linear length in a connected graph by using any spanning tree, and traversing an Euler tour of the tree twice and using the edges in the traversal in order. Thus, we can guess the length L at the very beginning in the range $[1, 4(n-1)]$.

2.1 Algorithm

Our algorithm exploits two main observations that were introduced by Iglesias, Rajaraman, Ravi and Sundaram [6].

2.1.1 Idea 1

The first key idea is the following. If there are numerous sources that want to send to the same sink, we can satisfy **all demands** that originate in those sources. Algorithm 1 shows how this can be done. The analysis of the algorithm is summarized in Lemma 3.

► **Lemma 3** (Big in-demand [6]). *Given a vertex t that is a sink in demand pairs $(s_1, t), \dots, (s_d, t)$, Algorithm 1 satisfies **all demands** from nodes s_1, \dots, s_d in $\tilde{O}(L)$ steps.*

Note that we are not only satisfying d demand pairs, but potentially up to $d \cdot n$ demands, because every s_i could be a source of up to n demand pairs.

■ **Algorithm 1** Algorithm for satisfying all demands in the “demand-neighbourhood” of a given vertex.

procedure INDEMAND(G, S, t)

Input: graph G , source nodes $S = \{s_1, \dots, s_d\}$ with the same sink t .

Run TELEPHONEMULTICAST(G, t, S) to get schedule TM that spreads information from t to all nodes in S under telephone model.

Reverse schedule TM and run it on G to collect all information from s_1, \dots, s_d in t .

Run BFS(t) to spread information from t to all sinks that correspond to sources in S .

end procedure

2.1.2 Idea 2

The key observation here is the following. If we can find a lot of disjoint source-destination paths, we can satisfy all of them **in parallel**, and hence in at most the length of the longest path. We will try to greedily find as many paths of length at most L as possible from given sources to corresponding destinations. Algorithm 2 summarizes this procedure.

■ **Algorithm 2** Algorithm for finding disjoint paths between sources and destinations.

procedure DISJOINTPATH(G, D, L)

Input: graph G , demand pairs $D = \{(s_1, t_1), \dots, (s_p, t_p)\}$ and the length of the optimal algorithm L .

$P = \emptyset$

while there is a demand pair $(s, t) \in D$ such that the distance between s and t is at most L **do**

 Add shortest path from s to t to P .

 Delete all vertices in this path from G .

end while

Return P .

end procedure

2.1.3 Our idea

Iglesias, Rajaraman, Ravi and Sundaram [6] used both ideas and balanced parameters in them to achieve $\tilde{O}(n^{\frac{2}{3}})$ -approximation. We will combine and generalize these ideas in a different way.

Idea 2.1.1 is useful when there is a vertex with high in-demand. More precisely, it allows us to reduce the number of distinct sources while there are a lot of demand pairs.

Idea 2.1.2 instead shows how we can either satisfy a lot of demand pairs in parallel or significantly reduce the number of distinct sources.

Our contribution comes from generalizing both ideas and combining them alternatively. We use the first idea to reduce the number of demand pairs, then we use the second idea to reduce the number of distinct sources. Now, given smaller number of sources we exploit the first idea again to reduce the number of demand pairs even further, etc. Algorithm 4 corresponds to the first idea of reducing the number of demand pairs given the number of distinct sources, whereas Algorithm 5 corresponds to the second idea of reducing the number of distinct sources given fixed number of demand pairs. These two algorithms call each other recursively. If we keep doing this infinitely long, we will achieve $\tilde{O}(\sqrt{n})$ -approximation, so the main Algorithm 3 stops when it gets ε -close to it.

The analysis of the algorithm is presented in the main Lemma 7. The key idea in the analysis is that the first observation can improve the performance of the second one, and vice versa. So we alternatively apply each observation to improve the previous one.

2.2 Correctness and Complexity

▷ **Claim 4.** Algorithm 3 satisfies all given demands in the graph under wireless model rules.

First, note that the length L of the optimal schedule is at most $4(n - 1)$, where n is the number of vertices, so one of the runs of $S(G, D, k, L)$ will use the correct guess of L and hence will satisfy all demands as long as Algorithm 4 works properly.

▷ **Claim 5.** Algorithms 4 and 5 satisfies all given demands in the graph under wireless model rules.

Proof. First, note that algorithms 4 and 5 call each other alternatively, but every second time, the parameter k (which keeps track of the number of iterations) decreases by 1, so it will eventually reach 0. Also, note that every time any of these algorithms modify the set of demand pairs D , they correctly satisfy or split demands. After k reaches 0, Algorithm 4 just runs BFS from every source node, which clearly satisfies all demands. ◁

■ **Algorithm 3** Algorithm for rumor spreading in the general graph.

```

procedure GENERALMM( $G, D, \varepsilon$ )
  Input: unweighted graph  $G$ , demand pairs  $D = \{(s_1, t_1), \dots, (s_p, t_p)\}$  and precision  $\varepsilon > 0$ .

  Take integer  $k = \lceil \frac{1}{4\varepsilon} \rceil$ .
  for  $L = 1, 2, \dots, 4(n-1)$  do           ▷ guess for the length of the optimal schedule
    if  $S(G, D, k, L)$  produces a shorter schedule then
      Save  $S(G, D, k, L)$  as current answer in BestSchedule.
    end if
  end for
  Return BestSchedule.
end procedure

```

■ **Algorithm 4** Algorithm for general graphs based on the number of sources.

```

procedure  $S(G, D, k, L)$ 
  Input: graph  $G$ , demand pairs  $D = \{(s_1, t_1), \dots, (s_p, t_p)\}$ , iteration number  $k$  and the length of the optimal schedule  $L$ .

  if  $k = 0$  then                               ▷ condition to exit the recursion
    for every distinct source  $s$  do
      Run BFS( $s$ ).                               ▷ to satisfy all demands in  $D$  that start in  $s$ 
    end for
    Exit.
  end if

  Compute  $s$  to be the number of vertices in the graph that are sources in at least one demand pair in  $D$ .
  Take  $\alpha = \frac{1+k \log_s n}{2k+1}$ .
  while there is a vertex  $t$  with at least  $s^{1-\alpha}$  demands going into it do
    Compute  $S_t$  to be those vertices that needs to send information to  $t$ . So  $S_t = \{s \mid (s, t) \in D\}$ .
    Run INDEMAND( $G, S_t, t$ ) to satisfy all demands that originate in  $S_t$ .
    Delete all demands that originate in  $S_t$  from  $D$ .
  end while
  Run  $P(G, D, k-1, L)$ .
end procedure

```

▷ **Claim 6.** Algorithm 3 runs in polynomial time($n, \frac{1}{\varepsilon}$).

Proof. First, note that Algorithm 1 runs in polynomial time, because TELEPHONEMULTICAST as well as BFS run in polynomial time. Since we can find shortest path between any two nodes in a graph in polynomial time, Greedy Algorithm 2 runs in polynomial time as well since we decrease the number of vertices in the graph at each iteration.

Then, Algorithm 3 runs Algorithm 4 $4(n-1)$ times, so it is polynomial time as long as Algorithm 4 is.

Algorithms 4 and 5 call each other alternatively, at most $k = \lceil \frac{1}{4\varepsilon} \rceil = O(\frac{1}{\varepsilon})$ times each. So it is enough to show that both of these algorithms each run in polynomial time.

■ **Algorithm 5** Algorithm for general graph based on the number of demand pairs.

procedure P(G, D, k, L)

Input: graph G , demand pairs $D = \{(s_1, t_1), \dots, (s_p, t_p)\}$ and iteration number k and the length of the optimal algorithm L .

Take $\beta = \frac{1+k \log_p n}{2k+2}$.

while the number of disjoint paths $d = |\text{DISJOINTPATH}(G, D, L)|$ is at least $p^{1-\beta}$ **do**

Satisfy all d demands in parallel following paths in $\text{DisjointPath}(G, D, L)$.

Delete all those demands from D .

end while

Denote $\text{DISJOINTPATH}(G, D, L)$ to be the disjoint paths for demand pairs $\{(s_1, t_1), \dots, (s_d, t_d)\}$.

Let a new graph G' be G with added complete binary tree with leaves t_1, \dots, t_d and root r .

Compute S to be all vertices in G that are sources in at least one demand pair in D .

$\text{TM} = \text{TELEPHONEMULTICAST}(G, r, S)$. \triangleright TM spreads information from the root r to all sources in D under telephone model

Reverse schedule TM and run it on G . \triangleright It will collect all information from sources in D into t_1, \dots, t_d

$D_{\text{new}} = \emptyset$

for $(s, t) \in D$ **do**

Find t_i among t_1, \dots, t_d that has information from source s .

Add demand pair (t_i, t) into D_{new} .

end for

Run $S(G, D_{\text{new}}, k, L)$.

end procedure

Algorithms 4 runs Algorithm 1 at most $\frac{s}{s^{1-\alpha}} = s^\alpha$ times. Given that $\alpha = \frac{1+k \log_s n}{2k+1}$ we have that $s^\alpha = \sqrt[2k+1]{sn^k} \leq \sqrt[2k+1]{n^{k+1}} \leq n$. As we have already noted, Algorithm 1 also runs in poly-time, so overall 4 is poly-time in terms of n and $\frac{1}{\varepsilon}$.

Algorithms 5 runs Algorithm 2 at most $\frac{p}{p^{1-\beta}} = p^\beta$ times. Given that $\beta = \frac{1+k \log_p n}{2k+2}$ we have that $p^\beta = \sqrt[2k+2]{pn^k} \leq \sqrt[2k+2]{n^{k+1}} \leq n$. As we have already noted, Algorithm 2 also runs in poly-time. So overall Algorithm 4 is polynomial time in terms of n and $\frac{1}{\varepsilon}$. \triangleleft

2.3 Analysis

First, assume that for the given graph and demand pairs the optimal schedule takes L time steps. The main Theorem 2 will follow from the following key lemma:

► **Lemma 7.** *For every $k \in \mathbb{N}_0$ the following holds:*

(S_k) *If the number of distinct sources is s , then there is a $\tilde{O}(\sqrt[2k+1]{sn^k})$ -approximation schedule.*

(P_k) *If the total number of demand pairs is p , then there is a $\tilde{O}(\sqrt[2k+2]{pn^k})$ -approximation schedule.*

Note that $s \leq n$, so for every $k \in \mathbb{N}_0$ from S_k we can have $\tilde{O}(\sqrt[2k+1]{sn^k}) = \tilde{O}(n^{\frac{k+1}{2k+1}})$ approximation schedule. As $k \rightarrow \infty$, $\frac{k+1}{2k+1} \rightarrow \frac{1}{2}$, and hence we can find a schedule that has approximation ratio $O(n^{\frac{1}{2}+\varepsilon})$ for any positive ε . Thus, Lemma 7 implies the main Theorem 2.

To prove the main lemma, we use an important observation that handles aggregated messages with a few sinks.

► **Lemma 8.** *If there is only one sink in multicommodity multicast under wireless model, then there is an $O(\frac{\log p}{\log \log p})$ -approximation algorithm for satisfying all demands, where p is the number of sources.*

Proof. The proof is based on the fact that collecting information from several sources to one sink t under wireless model is equivalent to multicasting information from t to those sources under telephone model (in reverse).

Note that because we are collecting information at one node, every piece of information travels on the path (the node never sends information to several neighbours at the same time) and so every step of the schedule is a matching - in other words, we have never used the power of wireless model over telephone model. Hence, collecting information at one node under wireless model is equivalent to collecting under telephone model, which is further equivalent to multicast under telephone model by reversing the schedule.

Using the known approximation result for multicasting under the telephone model given in Theorem 1, we can find a multicast schedule of length at most $O(L \cdot \frac{\log p}{\log \log p})$. Reversing it leads to a schedule that collects all information into t in time $O(L \cdot \frac{\log p}{\log \log p})$. ◀

We can now prove our initial lemma about satisfying all the demands of sources corresponding to a single sink.

Proof of the Lemma 3. We will produce a schedule that will satisfy all demands from nodes s_1, \dots, s_d in $\tilde{O}(L)$ steps. It will go in two stages:

1. Collect all information from s_1, \dots, s_d in t using Lemma 8.
2. Send it all together from t to all sinks of s_1, \dots, s_d using BFS.

Note that the second step could be done in $O(L)$ by BFS from t , because the distance from t to any sink of s_i is at most the distance from t to s_i plus the distance from s_i to sink, which is at most $L + L = 2L$. ◀

Proof of key Lemma 7. We prove this by induction on k . Moreover, the structure will be as follows.

1. $S_k \Rightarrow P_k$ for every $k \geq 0$.
2. $P_k \Rightarrow S_{k+1}$ for every $k \geq 0$.

Base case. It is enough to prove base case S_0 , in other words that it is possible to satisfy all demands with approximation ratio $O(s)$. Note that if we fix an arbitrary source v , we can satisfy all its demands in $O(L)$ steps by BFS starting at v (exploiting the power of wireless model where we can send to several neighbours at the same time). By repeating this over the s sources, it is possible to satisfy all demands in $O(L \cdot s)$ steps.

Induction step. Fix an arbitrary $k \geq 0$.

1. We want to prove that $S_k \Rightarrow P_k$.

To recall, this means that: if there is a $\tilde{O}(\sqrt[2^{k+1}]{sn^k})$ approximation schedule for s sources, then there is a $\tilde{O}(\sqrt[2^{k+2}]{pn^k})$ approximation schedule for p demand pairs.

Assume that we are given p demand pairs.

Forming greedy paths: Start with an arbitrary source and try to find a path of length at most L to one of its destinations. If found, delete all vertices in this path from the graph and repeat. This corresponds to Algorithm 2.

After this stage we have some maximal number d of disjoint paths $(s_1, t_1), \dots, (s_d, t_d)$. Note that because they are disjoint we can satisfy all of them in parallel in L steps. Let $\beta \in [0, 1]$ be a constant we will set afterwards. The idea of Algorithm 5 is the following.

- a. If the number of greedy paths d is at least $p^{1-\beta}$, then satisfy those demands in parallel and repeat.
- b. If the number of greedy paths d is less than $p^{1-\beta}$, we will reroute/collect all information at t_1, \dots, t_d using Lemma 9 and finish everything in one shot by applying S_k to it.

We cannot repeat the first case more than $\frac{p}{p^{1-\beta}} = p^\beta$ times, and so the total number of rounds in the schedule is at most $\tilde{O}(L \cdot p^\beta)$.

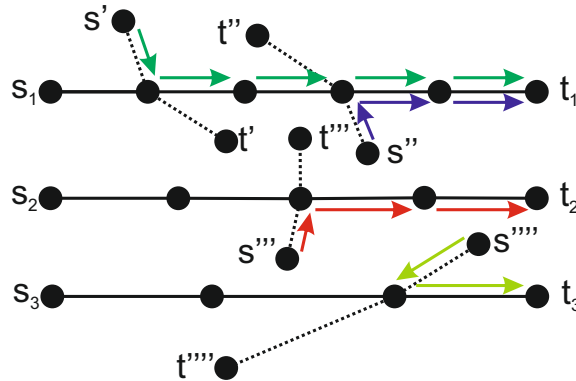
In the second case, there are $d \leq p^{1-\beta}$ sources now, so by applying S_k we get that we can send all information in $\tilde{O}(L \cdot {}^{2k+1}\sqrt{p^{1-\beta}n^k})$. Combining the two cases gives approximation ratio $\tilde{O}(p^\beta + {}^{2k+1}\sqrt{p^{1-\beta}n^k})$. Minimizing this leads to $\beta = \frac{1+k \log_p n}{2k+2}$ and approximation ratio $\tilde{O}({}^{2k+2}\sqrt{pn^k})$, which is exactly what we wanted.

Technical note: we need to ensure that $\beta \in [0, 1]$. Clearly $\beta \geq 0$. Now, $k = 0$ leads to $\beta = \frac{1}{2}$, giving a \sqrt{p} approximation ratio. For $k \geq 1$ we have that $p \geq n \Rightarrow \beta \leq 1$, but in the case $p < n$ the approximation ratio \sqrt{p} is better than $\tilde{O}({}^{2k+2}\sqrt{pn^k})$ for any k .

The only thing left is to prove the following lemma.

► **Lemma 9** ([6]). *If there are d greedy paths $(s_1, t_1), \dots, (s_d, t_d)$, we can collect all information at one of the t_1, \dots, t_d in $\tilde{O}(L)$ time.*

Figure 1 gives an example illustration of the Lemma 8.



■ **Figure 1** Collecting information from all sources into t_1, \dots, t_k .

Proof. For the simplicity of the argument (and to use results about broadcast time, which is the same as collecting information at one node), we will add to our graph a dummy complete binary tree with leaves t_1, \dots, t_d and root r and our goal will be to collect all information at r in $\tilde{O}(L)$ time. Obviously, if we can do this, then we also will be able to collect all information at one of the t_1, \dots, t_d in the same amount of time.

Consider an arbitrary source s and pick a fixed sink t for it (only one). Consider the path between $s \rightarrow t$ in the optimal schedule. This path has length at most L and hence it should intersect with path $s_i \rightarrow t_i$ for some i (otherwise we can find one more disjoint path of length at most L). This means that if we run the optimal schedule, then nodes on the greedy paths will contain information from all sources. Hence if we run it for L more steps by transmitting information only along greedy paths, information from all

sources will eventually be at one of t_1, \dots, t_d . The dummy binary tree has depth at most $\log d \leq \log n$, so if we run the schedule for $2 \log n$ more rounds (two parallel rounds per level), the root r will contain all the information.

This means that it is possible to collect all information in r in at most $2L + 2 \log n$ steps. Using Lemma 8 we can find schedule that collects all information into r in at most $O((2L + 2 \log n) \frac{\log n}{\log \log n}) = \tilde{O}(L)$ steps. ◀

2. We want to prove that $P_k \Rightarrow S_{k+1}$.

Here we will exploit Lemma 3. Note that we are not just satisfying d demand pairs, but potentially up to $d \cdot n$ demands, because every source s_i could be in up to n demand pairs. Also, by the base case we know that this could be done in $O(dL)$ steps, but this lemma gives a much better bound.

Now, assume that we have s sources, and let $\alpha \in [0, 1]$ be number that we will define afterwards. The main idea of Algorithm 4 is the following.

- a. **If there is a vertex with at least $s^{1-\alpha}$ demands** going into it, then satisfy those demands by Lemma 3 and repeat.
- b. **If every vertex has less than $s^{1-\alpha}$ demands** going into it, then the total number p of demand pairs is small and we will finish in one shot by applying P_k .

In the first case we can satisfy all demands in $\tilde{O}(L)$ steps by the lemma, and also we will repeat this at most $\frac{s}{s^{1-\alpha}} = s^\alpha$ times, so the total number of steps is $\tilde{O}(L \cdot s^\alpha)$.

In the second case, if every vertex has less than $s^{1-\alpha}$ demands going into it, then the total number of demands is $p \leq s^{1-\alpha}n$. Applying P_k gives that we can find a schedule to satisfy all demands in $\tilde{O}(L \cdot \sqrt[2k+2]{s^{1-\alpha}n^{k+1}})$.

Combining the two yields to approximation ratio $\tilde{O}(s^\alpha + \sqrt[2k+2]{s^{1-\alpha}n^{k+1}})$. By minimizing this we get $\alpha = \frac{1+(k+1)\log_s n}{2k+3}$ for the call to $S(G, D, k+1, L)$ and corresponding approximation ratio $\tilde{O}(\sqrt[2k+3]{sn^{k+1}})$, which is what we wanted.

Technical note: we need to ensure that $\alpha \in [0, 1]$. Clearly $\alpha \geq 0$. Note that $\alpha \leq 1$ if and only if $s \geq \sqrt{n}$, but in the case $s < \sqrt{n}$ the approximation ratio $O(s)$ in the base case is better than $\tilde{O}(\sqrt[2k+3]{sn^{k+1}})$ for any k . ◀

3 Reduction to packet routing

In this section we draw a connection between multicommodity multicast problem and extensively studied store-and-forward packet routing problem. We further derive an algorithm that satisfies all demands in $O(\text{OPT} + \text{number of demand pair})$ for multicommodity multicast under wireless model using approximation results from packet routing problem. Note that if the number of given demand pairs is small or the optimal schedule is long, this gives a better approximation result compared to the one we have shown in Section 2.

► **Theorem 10.** *There is a polynomial time algorithm that finds a schedule of length $O(L+p)$ for multicommodity multicast problem with p demand pairs under wireless model.*

3.1 Packet routing problem

We are going to relate wireless multicommodity multicast and packet routing problem.

► **Definition 11** (Store-and-forward packet routing problem). *Given an arbitrary (directed) graph G and source-sink pairs of vertices $(s_1, t_1), \dots, (s_p, t_p)$, where each source s_i has a unique packet, the problem is to find a schedule that transfers packets from every source s_i to every destination t_i in a minimum amount of steps, so that no two packets can traverse the same edge at the same unit of time.*

Note that under wireless model we are allowed to send several pieces of information at once along one edge, which is not the case for packet routing problem. On the other hand, packet routing problem has no other restrictions. This implies that a node can: (i) receive several packets at the same time (ii) send several packets as long as they use different edges (iii) send and receive at the same time.

The analysis of packet routing problem involves a trade-off between the “dilation” parameter d (maximum path length) and the “congestion” criterion c (maximum number of paths using any edge). Note that both c and d are lower bounds on the length of the optimal schedule for packet routing. Since the paths for the packets are not specified a priori, note that the problem also first involves finding such paths minimizing the sum of the dilation and congestion and then constructing an actual (offline centralized) schedule that completes in about this many rounds. Srinivasan and Teo [12] do just that and show how to construct a schedule of length $O(c + d)$, which achieves constant approximation.

► **Theorem 12** ([12]). *There is a polynomial time algorithm that finds a schedule for store-and-forward packet routing problem and gives a constant approximation.*

3.2 Reduction from wireless to packet routing

We prove Theorem 10 by reducing multicommodity multicast under the wireless model to packet routing problem and using Theorem 12. There are two superior properties in the packet routing problem: (i) a node can receive from several sources per time step; (ii) a node can send and receive at the same time step.

First, we will show how to eliminate (i). The number of incoming messages is a lower bound under wireless model (because a node can receive only from one source), whereas it is not the case in packet routing problem. Given an instance for wireless multicommodity multicast, we are going to modify it to create another instance for packet routing and use its schedule to design a fast wireless schedule that satisfies given demand pairs. Formally speaking, given an arbitrary undirected graph G we construct *directed* graph G' in the following way. For every node $v \in G$ we create two nodes v_{in} and v_{out} in G' and put a directed edge $v_{in} \rightarrow v_{out}$. Also, for every edge $(u, v) \in G$ we put directed edges $u_{out} \rightarrow v_{in}$ and $v_{out} \rightarrow u_{in}$ in G' . Then, every demand pair (s, t) in G correspond to demand pair (s_{in}, t_{out}) in G' . We split every original vertex into “only receiving” and “only sending” part. In this way the inbound restriction of v in G is translated into congestion restriction of edge $v_{in} \rightarrow v_{out}$ in G' .

We still need to address the second difference (ii) between packet routing and wireless. But for now we relax this condition and show how to transform wireless multicommodity multicast to packet routing allowing nodes to send and receive at the same time under wireless model.

► **Definition 13.** *The **Upgraded Wireless model** is the relaxation of the wireless model in which a node is allowed to send and receive information at the same time.*

► **Lemma 14** (Packet to information routing). *Given a schedule for packet routing problem on the graph G' , it is possible to transform it in polynomial time to a schedule of the same length on the graph G under the upgraded wireless model.*

Proof. First, given a packet routing schedule R on G' we modify R to schedule R' such that the following is true: (i) R' sends information through the same path as R (ii) R' finishes in the same amount of steps as R (iii) the graph G' does not contain any nodes that receive from more than one source at any time step. Then we show how given R' one can simply construct an upgraded wireless schedule of the same length on G .

Let's fix an arbitrary time step in R . Every node that sends to several destinations, or sends and receives at the same time can still do so under upgraded wireless model. The only issue is that a node cannot receive from several sources. Note that v_{out} can receive only from v_{in} , so we will show how to alter the schedule R so that every node v_{in} receives a packet from at most one neighbor. The main idea is that instead of accumulating packets at 'receiving' vertices v_{in} , we will accumulate information at 'sending' vertices u_{out} . More precisely, if vertex u_{out} sends packet p_i to v_{in} at some point t in time in R , then the new schedule R' does the following: vertex u_{out} will hold packet p_i and send it to v_{in} *right before* the time step when v_{in} sends p_i to v_{out} in R . So we send information only when needed, otherwise hold it in the previous node.

Note that in R' there is no vertex that receives from more than one node at a time. Moreover, the time at which every packet arrives to any vertex v_{out} under R is the same as in R' . This follows from the fact that whenever some vertex v_{out} needs a packet p_i from v_{in} , vertex v_{in} would have already collected packet p_i by construction of R' .

We are left to show that packet routing R' on graph G' could be transformed into upgraded wireless schedule W of the same length on graph G . Every time vertex u_{out} sends a packet p_i to v_{in} , let W send information i from u to v . Indeed, W has the same number of steps as R' and satisfies all demands, because it just mirrors packet schedule of R' . Moreover, W is a valid upgraded wireless schedule, since in R' every node receives from at most one node at every time step. ◀

► **Lemma 15.** *Given a schedule under upgraded wireless model it is possible to transform it in polynomial time to a schedule under wireless model, which has at most 3 times as many steps.*

Proof. Fix a time step and let S to be the directed graph of information flow of the given schedule under upgraded wireless model at this specific time step. So there is a directed edge $u \rightarrow v$ in S if and only if node u sends information to node v during this time step. We will show how to send all information that S does using wireless model in three time steps.

Note that the in-degree of every node in S is at most 1, because under upgraded wireless model a node still cannot receive from more than one source per time step. Each connected component in any directed graph with in-degree bounded by 1 has a simple characterization. It is a directed cycle with outbound trees hanging from it. More precisely, it is a union of the cycle $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k \rightarrow v_1$ with several trees each rooted at one of v_i 's, where every edge in a tree is directed out from the root. Note that it is enough to show how to transfer information in one connected component of schedule S using the wireless model and then run this in parallel across different connected components.

Deleting an edge $v_k \rightarrow v_1$ from the cycle (any other edge works as well) leads us to a tree rooted at v_1 , where every edge is directed out from the root. We show how to send all information along the edges in such a tree. We apply the following trick: let the *level* of the vertex be its distance to the root of the tree. Now, call the vertex *even* if its level is even and *odd* if its level is odd. First, all odd nodes that need to send information will do so (and so even ones will be able to receive). Then, all even nodes will be allowed to send information (and so odd ones will receive). In this way every node is either receiving or sending information. Finally, we send information along the deleted edge $v_k \rightarrow v_1$ from the cycle. Hence instead of sending all information in one time step, we did it in three. This blows up the total number of time steps by a factor of 3. ◀

► **Remark.** The result in Lemma 15 is sharp, for instance when underlying demand graph S is a directed cycle on 3 vertices. Under thre upgraded wireless model information could be sent in 1 step, whereas under the usual wireless model it requires 3 steps.

We are going to combine these results to prove Theorem 10. Given multicommodity multicast problem under wireless model on graph G with p demand pairs, we construct graph G' as described. Also, for every demand pair (s_i, t_i) in G we create a separate packet that originates at s_i and needs to be delivered to t_i . Note that even if there is one piece of information that originates at source s and needs to be sent to different sinks t_1, \dots, t_k , we create *different* packets p_1, \dots, p_k for each of these demands. If we consider optimal wireless schedule of length L on G and trace the path of every packet p_i , then the length of every such path is at most L , and on every edge the number of paths that uses that edge is at most p (because there are p paths in total). Hence, dilation $d \leq L$ and congestion $c \leq p$. We do not know the optimal wireless schedule, but both c and d are lower bound on the length of optimal packet routing schedule, and hence $\frac{c+d}{2}$ is also a lower bound. Theorem 12 gives a constant-approximation schedule for packet routing, and hence this schedule satisfy all demands in $O(c + d) = O(L + p)$ time steps. Then we exploit Lemma 14 and Lemma 15 to transform it into a $O(L + p)$ schedule for multicommodity multicast problem under the wireless model, which proves Theorem 10.

4 Rumor spreading in trees

In this section we present a simple polynomial algorithm TREE that gives a constant approximation for wireless rumor spreading in trees, which improves the best known $O(\frac{\log n}{\log \log n})$ -approximation ratio for n -node trees.

► **Theorem 16.** *Algorithm TREE runs in polynomial time and gives 3-approximation for multicommodity multicast problem in a tree under wireless model.*

The algorithm consists of two parts. We use the same idea as Iglesias, Rajaraman, Ravi and Sundaram [6], by rooting the tree and splitting rumor spreading into sending all information *up the tree* to ancestors, and then sending it *down the tree* to descendants.

4.1 Algorithm

■ **Algorithm 6** Algorithm for rumor spreading in the tree.

```

procedure TREE( $T, D$ )
  Input: Tree  $T$  and demand pairs  $D = \{(s_1, t_1), \dots, (s_p, t_p)\}$ .
  Pick an arbitrary vertex  $r$  and root the tree  $T$  at  $r$ .
   $D_{up} := \emptyset$ 
   $D_{down} := \emptyset$ 
  for  $(s, t) \in D$  do
    Find the least common ancestor  $lca(s, t)$  of  $s$  and  $t$ .
    Add demand pair  $(s, lca(s, t))$  to  $D_{up}$ .
    Add demand pair  $(lca(s, t), t)$  to  $D_{down}$ .
  end for
  Run TreeUp( $T, D_{up}$ ).
  Run TreeDown( $T, D_{down}$ ).
end procedure

```

In order to send information *up* the tree, algorithm TREEUP uses a greedy approach to send information that should go the furthest first. This strategy satisfies all demand pairs in at most $2 \cdot \text{OPT}$ time steps, where OPT is the length of the optimal schedule. Lemma 19 provides an analysis of the algorithm.

■ **Algorithm 7** Algorithm for rumor spreading, when all demands go up the tree.

```

procedure TREEUP( $T, D$ )
  Input: Rooted tree  $T$  and demand pairs  $D = \{(s_1, t_1), \dots, (s_p, t_p)\}$ , where  $t_i$  is an
  ancestor of  $s_i$  in the tree, for every  $i$ .
   $D_{clean} := \emptyset$  ▷ This will contain a subset of demand pairs  $D$  with non-repeating sources.
  for every vertex  $s \in G$  that is a source in  $D$  do
    Among all demand pairs in  $D$  with source  $s$  consider the one, which has the longest
    distance between  $s$  and corresponding sink in the tree.
    Add this pair in  $D_{clean}$ .
  end for
  Set the depth of every node  $v$ ,  $d(v)$  to be the length of the path from  $v$  to the root  $r$ .
  Set current number of steps  $i := 0$ .
  while not all demands in  $D_{clean}$  are satisfied do
    for  $v \in G$  do
      if  $d(v)$  has same parity as  $i$  then
        Find child  $u$  of  $v$  with the piece of information that at the current state
        needs to travel the longest distance up in the tree to its sink.
        Send all information  $u$  has up to  $v$ .
      end if
    end for
     $i = i + 1$ 
  end while
end procedure

```

The algorithm TREEDOWN sends information *down* the tree very fast, using two observations. First, it exploits the fact that under wireless model information could be sent to several neighbours simultaneously. Second, we note that any two nodes in a tree have disjoint set of children. We show that this strategy shows satisfies all demands in at most $\text{OPT}+1$ steps. Lemma 20 gives this analysis of algorithm TREEDOWN 8.

■ **Algorithm 8** Algorithm for rumor spreading, when all demands go down the tree.

```

procedure TREEDOWN( $T, D$ )
  Input: Rooted tree  $T$  and demand pairs  $D = \{(s_1, t_1), \dots, (s_p, t_p)\}$ , where  $t_i$  is a
  descendant of  $s_i$  in the tree, for every  $i$ .
  Set the depth of every node  $v$ ,  $d(v)$  to be the length of the path from  $v$  to the root  $r$ .
  Set current number of steps  $i := 0$ .
  while not all demands are satisfied do
    for  $v \in G$  do
      if  $d(v)$  has same parity as  $i$  then
        send all information  $v$  has to all its children in the tree.
      end if
    end for
     $i = i + 1$ 
  end while
end procedure

```

Combining these two algorithms, we get the overall algorithm TREE for wireless multicommodity multicast in the tree.

4.2 Correctness and Complexity

▷ **Claim 17.** Algorithm TREE 6 satisfies all given demands in the tree under wireless model rules.

Proof. First, it is clear that if we first send information from source s to $lca(s, t)$ for every demand pair (s, t) , and then from $lca(s, t)$ to t for all (s, t) , then we would have sent information from every s to corresponding t .

Now, we need to show that algorithms TREEUP 7 and TREEDOWN 8 correctly send information up and down the tree respectively. First, let's show that both algorithms obey wireless rules. Note that in both algorithms only nodes with the same parity of depth can send information, and so node with opposite parity of depth will receive information, hence there is no vertex that is trying to send and receive information at the same time.

Next, we show that none of the nodes is trying to receive information from more than one vertex. Every vertex v in algorithm TREEUP 7 receives from only one of its children (if any). For algorithm TREEDOWN 8, observe that nodes with even (or odd) depth are pairwise not connected, and so sets of their children are disjoint.

Finally, note that both algorithms will run until all demands are satisfied. Since at every step at least one piece of information becomes closer to the destination, it should satisfy all demands in finite number of steps. ◁

▷ **Claim 18.** Algorithm TREE runs in polynomial time.

Proof. It is enough to show that both algorithm TREEUP 7 and algorithm TREEDOWN 8 run in polynomial time. Note that since at every step at least one piece of information becomes closer to the destination, the number of while loops is at most (number of demand pairs) · (maximum distance in the tree), which is polynomial in the number of vertices in the tree. In algorithm TREEUP 7 at each iteration of the while loop, we also need to loop through all children of v and find the one that needs to send information the furthest. This could be done in poly-time as well, because the number of children, pieces of information and diameter of the tree are all polynomial in n . ◁

We show that algorithm TREE outputs a rumor spreading schedule that is at most 3 times longer than the optimal one. Let us denote the length of the optimal schedule to be L .

It is enough to analyze approximation ratios of both algorithm TREEUP and algorithm TREEDOWN. We show that the first one gives a 2-approximation and the second one is almost optimal.

► **Lemma 19.** *Algorithm TREEUP 7 produces the schedule that satisfies all demands in at most $2L$ steps.*

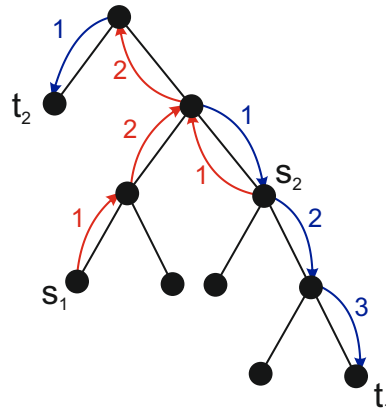
► **Lemma 20** ([6]). *Algorithm TREEDOWN 8 produces the schedule that satisfies all demands in at most $L + 1$ steps.*

These two lemmas clearly imply that algorithm TREE 6 has approximation ratio 3.

Proof of Lemma 19. Note that for every source s , all of its sinks could only be the vertices along the path from s to the root. Observe that for each $s - t$ pair the path from s to t is unique and hence we can also define a distance from s to t .

If s has to send to several sinks, we can leave only the furthest sink and remove others - because the tree path from s to the furthest sink also passes through closer sinks. So now we assume that every source has exactly one sink, which corresponds to the new set of demand pairs D_{clear} .

First, let us assume that the vertex is allowed to send and receive at the same time, but still is not allowed to receive from more than one source. Under this assumption, algorithm TREEUP 7 (after removing the line “If $d(v)$ has same parity as i ”) achieves the optimal spreading time! Figure 2 illustrates the execution of this modified version of algorithm TREEUP 7.



■ **Figure 2** Illustration of precursor to algorithm TREE 6.

Recall that the algorithm is the following: if node v contains some information, it will try to send it up. So if no other vertices are trying to send information to the parent of v , then v is allowed to send information and it is obviously *the best possible move*.

► **Definition 21** (The best/optimal move). *Given a multicommodity-multicast problem, the move is called optimal if there exists a subsequent sequence of moves that will achieve optimal rumor spreading time.*

Now, imagine that at the specific point in time there are several nodes v_1, \dots, v_k who want to send to their common parent t .

► **Definition 22** (The priority of the vertex). *Given a vertex v that has information from sources s_1, \dots, s_k , define the priority of v at this particular state of the system to be the maximum distance from v to all sinks t_1, \dots, t_k that correspond to sources s_1, \dots, s_k .*

The rule will be that vertices with larger priorities always send first. More precisely, if there are several nodes v_1, \dots, v_k who want to send information to their common parent t at this point in time, then pick the vertex v_i with the largest priority and allow it to propagate information to t (we will break any ties arbitrarily).

We argue that this is an optimal move. Consider the optimal algorithm and the first moment in time when the state is different from the state in our greedy algorithm. We will show that optimal algorithm could be modified, so that it still finishes rumor spreading in the same number of steps, but also does the same move as our greedy algorithm.

Assume at this state vertices v_1, \dots, v_k want to send to their parent t . One of the vertices v_i should be allowed to send info to t since one more piece of information will move closer to the sink this way. Consider what the optimal algorithm will do. Say it allows vertex v_j to send information to its parent. If $v_i = v_j$, then we are done (meaning our current move is optimal). If not, then consider a different algorithm, where we swap v_i in place of v_j , and move this step where v_j sends information to its parent to the next time when v_i is scheduled to do so in the optimal algorithm. We claim that this modified algorithm is still optimal.

To see this, consider the optimal algorithm again. Both vertices v_i and v_j have common path to the top (except the very first edge to their parent, which we will ignore). Moreover, because the priority of v_i is at least the priority of v_j , then the demand from v_j is a subpath of the demand for v_i (remember we associate demand with both the source-sink pair and the path between them). Because v_j goes first and takes the same route as v_i , but v_i goes further, we have that at some point during the optimal schedule the current information from vertices v_i and v_j will be at the same vertex (somewhere along the demand path). Hence at the end of the optimal schedule both sinks that correspond to v_i and v_j will contain the current information from both v_i, v_j . And so if we swap v_i and v_j in our algorithm, after the same number of time steps as in the optimal schedule again, both sinks that correspond to v_i, v_j will contain current information from both v_i, v_j . Hence our modified algorithm is optimal as well.

To sum up, this proves that our greedy algorithm which allows vertices with higher priority to send first is optimal.

To finish the proof we need to get rid of the assumption that a node can send and receive information at the same time. For this, we use the following idea: let the *level* of the vertex be its distance to the root of the tree. Now, call the vertex *even* if its level is even and *odd* if its level is odd. We will alternate between even/odd levels to send/receive. First, all odd nodes that need to send information will do so (and so even ones will be able to receive). Then, all even nodes will be allowed to send information (and so odd ones will receive). This is then repeated. In this way every node is either receiving or sending information at every step. Clearly, we have increased the number of time step by at most factor 2. Therefore our algorithm will finish in at most $2L$ steps. ◀

Proof of Lemma 20. First, note that the distance between any source and sink is at most L , because in the optimal schedule all distances are at most L . But we cannot start greedily send information down, because a vertex cannot both send and receive information at the same time.

Hence we will use the same idea again - vertices with even/odd levels will alternate to send/receive. Note that our algorithm TREEDOWN 8 does the following: first, all even nodes that need to send information will do so (and so odd ones will be able to receive). Then, all odd nodes will be allowed to send information (and so even ones will receive). This is repeated as before.

Consider a fixed source. If it is even, then it will start sending information down right away, if it is odd, there will be a delay of one step. But after the vertex started sending information, there will be no delays! For example, if the vertex is even, then it sends information to the odd one, and at the next iteration all odd vertices are allowed to send information. Hence, it will take at most $L + 1$ steps to deliver all information. ◀

4.3 Optimal Telephone Broadcast in a Tree

► **Corollary 23** (Optimal broadcast in the tree under telephone model). *There is a **simple** optimal schedule for broadcasting in the tree from any root under telephone model.*

Note that it is well-known that the optimal schedule for broadcast in the tree under telephone model can be found in polynomial time. For instance, one can find such a schedule using dynamic programming. But this approach only gives an implicit algorithm to find such a schedule. We will present a very **simple explicit optimal schedule** algorithm TREEBROADCAST 9.

■ **Algorithm 9** Algorithm for optimal broadcast in the tree under telephone model.

```

procedure TREEBROADCAST( $T$ )
  Input: rooted tree  $T$ .

  OptimalSchedule =  $\emptyset$ 
  while  $T$  is not empty do
    Matching =  $\emptyset$ 
    for every vertex  $v \in T$  do
      If  $v$  is not connected to any leaf, then continue to another  $v$ .
      Among all children of  $v$ , pick an arbitrary leaf  $u$ .
      Add edge  $u \rightarrow v$  to Matching.
    end for
    Add Matching to OptimalSchedule.
    Delete Matching from  $T$  and any isolated vertices.
  end while

  Return reversed OptimalSchedule
end procedure

```

▷ **Claim 24.** Algorithm TREEBROADCAST 9 outputs the optimal broadcast schedule in the tree.

Proof. Every broadcast schedule that sends information from the root to all nodes is equivalent to a schedule that sends information from the root to all leaves. This is further equivalent to a schedule that collects information from all leaves to the root - by reversing the direction of information flow and hence the schedule. We will present such a schedule.

Assume we are given that each leaf has a unique piece of information that it wants to transmit to the root. The schedule is very simple: every leaf that has some information (one or several pieces collected from other nodes) tries to send it to its parent. After the leaf successfully propagates the information up, we delete this vertex from the tree. If there are several leaves who try to send to the same parent, then choose one arbitrarily.

This is an optimal move because in each conflict when several leaves are trying to send to the parent, the priority of each of these conflicting leaves is the same (since their distance to the root is the same), so the argument is exactly the same as in the analysis of Algorithm TREEUP.

The only additional point here is that we do not allow for non-leaf vertices to send information, even though they might have some information available. This is because if there is an internal node v , such that in its subtree there are leaves with information that they haven't sent up to v yet, there is no need in an optimal scheme for v to try to send available information up, since it can wait until the very last information in its subtree will reach v . More formally, the priority of the node v is less than the priority of a leaf in a subtree of v , and hence we should give a priority to the leaf (by the same argument as in that of Algorithm TREEUP). This rule makes sure that there is no vertex that is trying to send and receive information and the same time. ◁

5 Conclusion

We have studied well-known rumor spreading problem under the wireless model. We designed approximation algorithms for the most general multicommodity multicast set-up that improve approximation ratios both for general graphs and when the underlying graph is a tree. For

general graphs we improve approximation ratio for this problem from $\tilde{O}(n^{\frac{2}{3}})$ to $\tilde{O}(n^{\frac{1}{2}+\epsilon})$ on n -node graphs. We also design an algorithm that satisfies p demand pairs in $O(\text{OPT} + p)$ steps, by reducing it to the well-studied packet routing problem. When underlying graph is an n -node tree, we improved approximation ratio from $O(\frac{\log n}{\log \log n})$ to a constant 3-approximation. A consequence of our algorithm is the simple constructive rule for optimal broadcasting in a tree under a widely studied telephone model.

References

- 1 Marek Chrobak, Leszek Gaasieniec, and Wojciech Rytter. Fast broadcasting and gossiping in radio networks. *Journal of Algorithms*, pages 177–189, 2002. doi:10.1016/S0196-6774(02)00004-4.
- 2 Michael Elkin and Guy Kortsarz. Sublogarithmic Approximation for Telephone Multiast. *SODA Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 76–85, 2003.
- 3 Arthur M. Farley. Broadcast Time in Communication Networks. *SIAM J. Appl. Math.*, pages 385–390, 1980. doi:10.1137/0139032.
- 4 András Hajnal, Eric C Milner, and Endre Szemerédi. A cure for the telephone disease. *Canadian Mathematical Bulletin*, 15(3):447–450, 1972.
- 5 Sandra M. Hedetniemi, Stephen T. Hedetniemi, and Arthur L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, pages 319–349, 1988. doi:10.1002/net.3230180406.
- 6 Jennifer Iglesias, Rajmohan Rajaraman, R Ravi, and Ravi Sundaram. Rumors across radio, wireless, telephone. In *35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
- 7 Jennifer Iglesias, Rajmohan Rajaraman, R Ravi, and Ravi Sundaram. Plane gossip: Approximating rumor spread in planar graphs. In *Latin American Symposium on Theoretical Informatics*, pages 611–624. Springer, 2018.
- 8 Tom Leighton, Bruce Maggs, and Satish Rao. Universal packet routing algorithms. In *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pages 256–269. IEEE, 1988.
- 9 Afshin Nikzad and R Ravi. Sending secrets swiftly: Approximation algorithms for generalized multicast problems. In *International Colloquium on Automata, Languages, and Programming*, pages 568–607. Springer, 2014.
- 10 R Ravi. Rapid rumor ramification: Approximating the minimum broadcast time. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 202–213. IEEE, 1994.
- 11 Daniel J. Rosenkrantz, Richard E. Stearns, and Philip M. Lewis II. System level concurrency control for distributed database systems. *ACM Transactions on Database Systems (TODS)*, pages 178–198, 1978.
- 12 Aravind Srinivasan and Chung-Piaw Teo. A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria. In *STOC '97 Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 636–643, 1997. doi:10.1145/258533.258658.

Recognizing Planar Laman Graphs

Jonathan Rollin

Theoretische Informatik, FernUniversität in Hagen, Hagen, Germany
jonathan.rollin@fernuni-hagen.de

Lena Schlipf

Theoretische Informatik, FernUniversität in Hagen, Hagen, Germany
lena.schlipf@fernuni-hagen.de

André Schulz

Theoretische Informatik, FernUniversität in Hagen, Hagen, Germany
andre.schulz@fernuni-hagen.de

Abstract

Laman graphs are the minimally rigid graphs in the plane. We present two algorithms for recognizing planar Laman graphs. A simple algorithm with running time $O(n^{3/2})$ and a more complicated algorithm with running time $O(n \log^3 n)$ based on involved planar network flow algorithms. Both improve upon the previously fastest algorithm for general graphs by Gabow and Westermann [*Algorithmica*, 7(5-6):465–497, 1992] with running time $O(n\sqrt{n \log n})$.

To solve this problem we introduce two algorithms (with the running times stated above) that check whether for a directed planar graph G , disjoint sets $S, T \subseteq V(G)$, and a fixed k the following connectivity condition holds: for each vertex $s \in S$ there are k directed paths from s to T pairwise having only vertex s in common. This variant of connectivity seems interesting on its own.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases planar graphs, Laman graphs, network flow, connectivity

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.79

Related Version A preliminary version of this article appeared at EuroCG 2019 (available from <http://www.eurocg2019.uu.nl/>).

1 Introduction

Let $G = (V, E)$ be a graph with n vertices. The graph G is called a *Laman graph* if it has $2n - 3$ edges and every subset $V' \subseteq V$, with $|V'| \geq 2$, induces a subgraph with no more than $2|V'| - 3$ edges. A *bar-joint framework* is a physical structure made from fixed-length bars that are linked by universal joints (allowing 360° rotations) at their endpoints. A bar-joint framework is *flexible* if it has a continuous motion other than a global rotation or translation. A nonflexible framework is called *rigid*. Moreover it is called *minimally rigid*, if it is rigid, but it becomes flexible after removing any bar. For detailed definitions we refer to Jordán [21]. Interestingly, in 2d a bar-joint framework (in a generic configuration) is minimally rigid, if and only if its underlying graph is a Laman graph. In other words, almost every embedding of a Laman graph will be rigid. For nongeneric configurations Laman graphs can yield a flexible framework though (see Figure 1).

Various characterizations of Laman graphs are known [17, 25, 26]. The class of *plane* Laman graphs provides even more structure [9, 16, 23]. Of particular interest for our result is the following geometric characterization: A geometric graph is a *pointed pseudotriangulation* (PPT) if each inner face contains exactly three angles less than π , called *small*, and every vertex is incident to an angle larger than π , called *big* [31]. Streinu [33] proved that the underlying graph of a pointed pseudotriangulation is a Laman graph. Moreover, Haas et al. [16] showed that every planar Laman graph has an embedding as a pointed pseudotriangulation. Pointed



© Jonathan Rollin, Lena Schlipf, and André Schulz;
licensed under Creative Commons License CC-BY

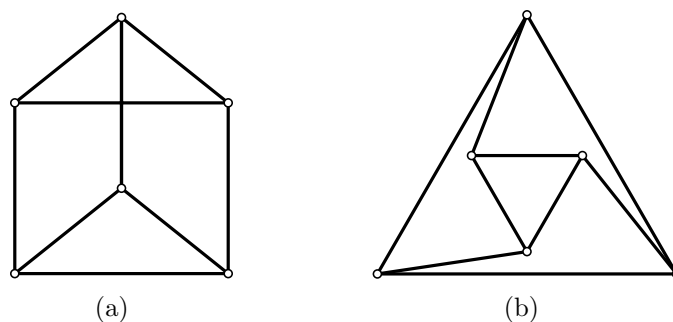
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 79; pp. 79:1–79:12

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** A flexible drawing of a Laman graph (a), and a rigid drawing of the same graph (b).

pseudotriangulations have many applications as a geometric data structure. Maybe most prominently, they are used as a tool to show that every polygonal linkage can unfold in 2d without intersections [6, 33]. More applications come from problems in ray shooting, motion planning and polygon guarding. We refer the interested reader to the survey of Rote, Santos, and Streinu [31].

Haas et al. [16] also give an algorithm that computes a PPT realization for a Laman graph in time $O(n^{3/2})$. We could turn their algorithm into a recognition algorithm by checking whether the derived realization is a PPT. If the original graph is not a Laman graph, then the construction either fails at some intermediate step or some parts of the drawing collapse. To check whether it collapsed requires computations with exponentially large numbers. So it depends on the model of computation if the overall algorithm runs in $O(n^{3/2})$ time. We present combinatorial algorithms for recognizing planar Laman avoiding these subtleties.

Checking the Laman condition for general graphs can be done in polynomial time. The fastest (but very complicated) algorithm is due to Gabow and Westermann [13] and needs $O(n\sqrt{n\log n})$ time. Their algorithm is based on a characterization by means of matroid sums (see also [7] for a time analysis specifically for Laman graphs). There is also a very easy so-called pebble-game algorithm that runs in $O(n^2)$ time [3, 20] (a brief survey of further algorithmic aspects is given in the introduction of [26]).

1.1 Our contribution

Our recognition algorithms for planar Laman graphs rely on the theory of combinatorial pseudotriangulations developed by Haas et al. [16] which will be described in detail in Section 4. The crucial step in the algorithm is to check the following connectivity condition for a certain auxiliary directed planar graph. Consider a positive integer k , a directed graph G , and disjoint sets $S, T \subseteq V(G)$. We call S k -connected to T if for each vertex $s \in S$ there are k directed paths from s to T pairwise having only vertex s in common.

A naive approach to check this condition can be implemented in $O(|S|m)$ time and is described in Section 1.2. We present two algorithms checking this condition for planar directed graphs that are faster for large sets S . In Section 2 we present a simple algorithm that checks this connectivity condition for planar directed graphs provided that each vertex is either in S or T . This gives the following theorem.

► **Theorem 1.** *For each fixed $k \geq 1$ there is an algorithm deciding for a directed planar graph G and a partition $V(G) = S \cup T$ whether S is k -connected to T in $O(n^{3/2})$ time.*

We show how to combine this algorithm with simple algorithms of Haas et al. [16] to check the Laman property for a plane graph in time $O(n^{3/2})$ in Section 4.

If all vertices in T are incident to a common face then we obtain a much faster algorithm for the connectivity check. We then even could drop the condition that each vertex is either in S or T . It is based on latest planar network flow algorithms and presented in Section 3.

► **Theorem 2.** *For each fixed $k \geq 1$ there is an algorithm deciding for a directed planar graph G and disjoint sets $S, T \subseteq V(G)$ whether S is k -connected to T in $O(n \log^3 n)$ time, provided that all vertices in T are incident to a common face in G .*

It turns out that this condition is satisfied when checking for the Laman property. This leads to a fast algorithm for recognizing planar Laman graphs which we describe in Section 4.

► **Theorem 3.** *There is an algorithm deciding for each planar graph whether it is a Laman graph in $O(n \log^3 n)$ time.*

1.2 Related work on connectivity in directed graphs

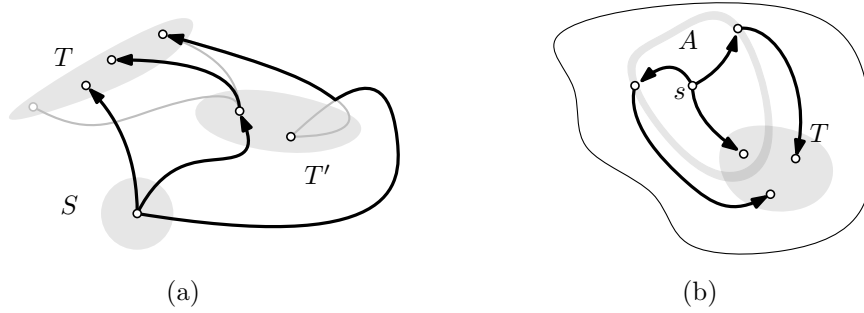
Here we discuss how the algorithms from Theorems 1 and 2 relate to known algorithms. We briefly recall the basic definitions and results on connectivity of directed graphs first. Note that we consider only unweighted graphs. For two vertices s and t in a directed graph let $\kappa(s, t)$ and $\lambda(s, t)$ denote the largest number of internally vertex disjoint respectively edge disjoint directed paths from s to t . It is well known that $\lambda(s, t)$ is equal to the value of a maximum s - t flow and hence to the size of a minimum s - t cut. A directed graph is strongly connected if for each vertex there is a directed path to each other vertex. A directed graph with at least k vertices is called k -vertex connected (k -edge connected) if the removal of any $k - 1$ vertices (edges) yields a strongly connected directed graph. Let $\kappa(G)$ ($\lambda(G)$) denote the largest k such that G is k -vertex connected (k -edge connected). By means of Menger's theorem we have that $\kappa(G) = \min_{s \neq t} \kappa(s, t)$ and $\lambda(G) = \min_{s \neq t} \lambda(s, t)$ [2].

For general directed graphs $\kappa(s, t)$ and $\lambda(s, t)$ can be computed with time in $O(nm)$ [30]. Checking whether $\kappa(s, t) \geq k$ or whether $\lambda(s, t) \geq k$ for some fixed k can be done by k steps of the Ford–Fulkerson algorithm and hence in $O(m)$ time (see [2, p. 205]). In both cases the computation of vertex connectivity can be easily reduced to the edge connectivity [10]. Further the values of $\kappa(G)$ and $\lambda(G)$ can be computed with algorithms due to Gabow with time in $O(m(n + \min(\kappa(G)^{5/2}, \kappa(G)n^{3/4})))$ [12] and time in $O(\lambda(G)m \log(n^2/m))$ [11] respectively. Checking whether $\kappa(G) \geq 2$ and whether $\lambda(G) \geq 2$ can be done with time in $O(n + m)$ [14, 19]. For larger fixed k the fastest algorithms checking $\kappa(G) \geq k$ and $\lambda(G) \geq k$ are based on Gabow's algorithms above with times in $O(m(n + \min(k^{5/2}, kn^{3/4})))$ and $O(km \log(n^2/m))$ respectively.

For planar directed graphs $\kappa(s, t)$ and $\lambda(s, t)$ can be computed in linear time [8, 22] (and $O(n \log n)$ in the weighted case). Further $\lambda(G)$ can be calculated with time in $O(n \log \log n)$ [28]. We are not aware of algorithms computing $\kappa(G)$ specifically for planar directed graphs.

A naive approach to check whether a set S is k -connected to T (the notion considered in this paper) works as follows. First introduce a new vertex t and all arcs xt , $x \in T$, and then check whether $\kappa(s, t) \geq k$ for each $s \in S$. This leads to an $O(|S|m)$ time algorithm for general directed graphs and constant k . For planar directed graphs our algorithm from Theorem 1 is faster than this approach in case $|S| \in \omega(\sqrt{n})$ and $V(G) = S \cup T$. For planar directed graphs where the vertices in T can be embedded incident to a common face our algorithm from Theorem 2 is faster already for $|S| \in \omega(\log^3 n)$.

Further questions related to connectivity of (planar) directed graphs include the all-pairs reachability [15, 18], all pairs minimum cuts [4].



■ **Figure 2** If S is k -connected to $T \cup T'$ and T' is k -connected to T , then S is k -connected to T (a). This statement applied where $T' = A$ is a separator (b).

2 Checking directed k -connectivity

In this section we prove the statement of Theorem 1. We first give some structural results. The following statement is similar to Menger’s theorem. We provide a proof for completeness.

► **Lemma 4.** *Let $k \geq 0$, G be a directed (not necessarily planar) graph, and $S, T \subseteq V(G)$ be disjoint with $|T| \geq k$. Then S is k -connected to T if and only if for each $s \in S$ and $A \subseteq V(G) \setminus \{s\}$ with $|A| = k - 1$ there is a directed path from s to T not using the vertices in A .*

Proof. Clearly, if S is k -connected to T then for each $s \in S$ and $A \subseteq V(G) \setminus \{s\}$ with $|A| = k - 1$ there is a directed path from s to T not using the vertices in A . We deduce the reverse statement from Menger’s theorem. Obtain a directed graph G' by adding a new vertex z and edges tz for each $t \in T$. Then S is k -connected to T in G if and only if for each $s \in S$ there are k internally vertex disjoint directed paths joining s and z in G' . Suppose that for each $s \in S$ and $A \subseteq V(G) \setminus \{s\}$ with $|A| = k - 1$ there is a directed path from s to T in G not using the vertices in A . Then, under the same assumptions, there is a directed path from s to z in G' not using the vertices in A . So by Menger’s theorem there are k internally vertex disjoint directed paths joining s and z in G' . In particular S is k -connected to T in G . ◀

The following observation is crucial for using separators recursively. An illustration is given in Figure 2.

► **Lemma 5.** *Let G be a directed graph and let $S, T, T' \subseteq V(G)$ be disjoint. If S is k -connected to $T \cup T'$ and T' is k -connected to T , then S is k -connected to T .*

Proof. Let $s \in S$ and fix a set $A \subseteq V(G) \setminus \{s\}$ of size $k - 1$. We shall find a directed path from s to T not using vertices from A . There is a directed path from s to some vertex $u \in T \cup T'$ not using vertices from A since S is k -connected to $T \cup T'$. If $u \in T$ we are done. If $u \in T'$, then there is a directed path from u to T not using vertices from A since T' is k -connected to T . In each case there is a directed path from s to T not using vertices from A . So S is k -connected to T by Lemma 4. ◀

For a single vertex we can decide in linear time whether it is k -connected to T by means of Ford–Fulkerson’s algorithm (similar to the naive approach from Section 1.2). An st -connector in a directed graph G is a set of edge disjoint directed paths from vertex s to vertex t . Given an st -connector \mathcal{P} the residual graph $G_{\mathcal{P}}$ is obtained by reversing all arcs of paths in \mathcal{P} . It is well known that \mathcal{P} is a largest st -connector in G if and only if there is no directed path from s to t in $G_{\mathcal{P}}$ [2].

► **Lemma 6.** *For each $k \in \mathbb{N}$ there is an algorithm deciding for any directed (not necessarily planar) graph G , $s \in V(G)$, and $T \subseteq V(G)$ whether s is k -connected to T in $O(k(|V(G)| + |E(G)|))$ time.*

Proof. We apply the following standard modification due to Ford and Fulkerson [10]. Split each $v \in V(G)$ into two vertices v_i and v_o and replace each arc uv by an arc $u_o v_i$. Further add all arcs $v_i v_o$, $v \in V(G)$. Finally add a new vertex z and the arcs $t_o z$, $t \in T$. Let G' denote the resulting directed graph. Then s is k -connected to T in G if and only if there is an $s_o z$ -connector of size k in G' .

The algorithm iteratively builds an $s_o z$ -connector \mathcal{P} in G' in at most k steps. In each step a directed path P from s_o to z is searched in the residual graph $G'_\mathcal{P}$. If no such path exists, then the set \mathcal{P} is a largest $s_o z$ -connector in G' . Otherwise, in G' the symmetric difference of the set of arcs from paths in \mathcal{P} and the set of arcs from P (which might not be a path in G') forms an $s_o z$ -connector of size $|\mathcal{P}| + 1$ in G' .

Clearly the algorithm decides whether there is an $s_o z$ -connector of size k in G' and hence whether s is k -connected to T in G . The construction of G' in the beginning and in each step the construction of the residual graph as well as the search can be implemented in $O(n + m)$ time. Since there are at most k steps the overall time is $O(k(n + m))$. ◀

We call $A \subseteq V(G)$ a *separator* if removing A splits G into two (not necessarily connected) subgraphs G_1 and G_2 , such that $|V(G_1)|, |V(G_2)| \leq \frac{2}{3}|V(G)|$. For every planar graph G a separator with size in $O(\sqrt{n})$ can be found in linear time [27].

Proof of Theorem 1. The algorithm works recursively as follows. Let A denote a separator of G of size $O(\sqrt{n})$. Use Lemma 6 to check for each $a \in A \cap S$ whether a is k -connected to T in G . Let G_1 and G_2 denote the two subgraphs of G separated by A (each including A). For $i = 1, 2$ let $S_i = (S \cap V(G_i)) \setminus A$ and let $T_i = (T \cap V(G_i)) \cup A$. Apply the algorithm recursively to check whether S_i is k -connected to T_i in G_i , for $i = 1, 2$.

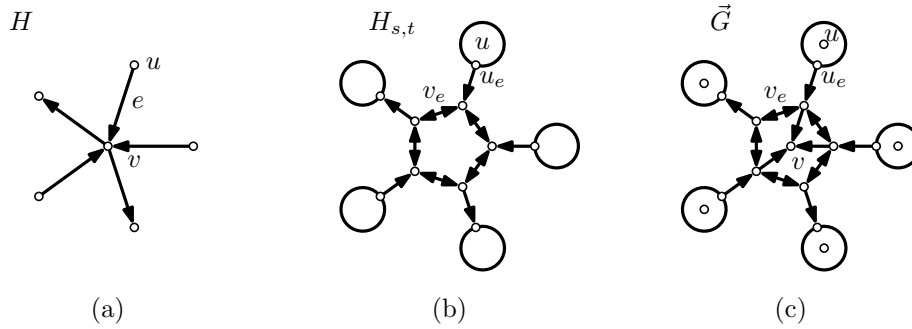
Recall that $V(G) = S \cup T$. The algorithm indeed checks whether S is k -connected to T in G since either it finds some vertex in $A \cap S$ that is not k -connected to T in G , or it is sufficient to check whether $S \setminus A$ is k -connected to $A \cup T$ by Lemma 5. Then it is sufficient to check G_1 and G_2 separately. See Figure 2(b) for an illustration

The separator can be found in linear time. Then the algorithm from Lemma 6 is called $O(\sqrt{n})$ times for each vertex in the separator, each call with linear time (since k is constant and $|E(G)|$ is linear in n). So the total time for each step of the recursion and hence for the whole algorithm is $O(n^{3/2})$. ◀

3 A faster algorithm for checking directed k -connectivity

In this section we prove Theorem 2. First we use a construction similar to one of Kaplan and Nussbaum [22] transforming the problem into a question about network flow. Consider a directed graph H and distinct $s, t \in V(H)$. Kaplan and Nussbaum construct a directed planar graph $H_{s,t}$ obtained from H by replacing each $v \in V(H) \setminus \{s, t\}$ by a cycle C_v with vertices v_1, \dots, v_d , where d is the degree of v in H , such that arcs incident to v are replaced by arcs of (flow) capacity 1 not sharing endpoints while keeping their orientation and the cyclic order around v . The edges of the cycle C_v are oriented in both directions and receive (flow) capacity $1/2$. See Figure 3.

► **Lemma 7** ([22]). *There are k internally vertex disjoint s - t -paths in H if and only if in $H_{s,t}$ the maximum s - t -flow has value at least k .*



■ **Figure 3** A directed planar graph H (a), the modification $H_{s,t}$ of Kaplan and Nussbaum (b), and the modification \vec{G} used in the algorithm for Theorem 2 (c). The modification of Kaplan and Nussbaum does not include the original vertices inside of the new cycles. We need this since all the original vertices except a fixed source are targets in our algorithm.

Since we need to calculate s - t -flows for several distinct targets t at the same time we need a modification independent of t .

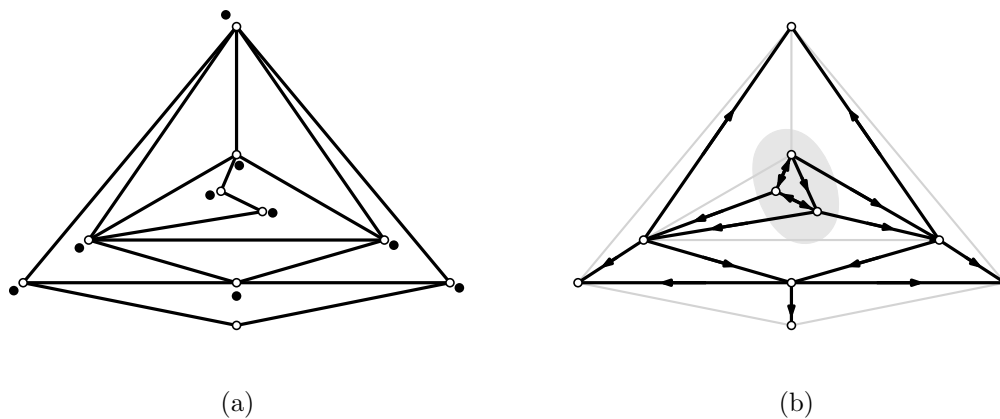
► **Lemma 8.** Consider $k \geq 1$, a directed planar graph G and disjoint sets $S, T \subseteq V(G)$ where all vertices of T are incident to a common face. Then a directed planar graph \vec{G} with $|V(\vec{G})| \leq 7|V(G)|$ can be computed in linear time together with some $s \in V(\vec{G})$ and $T' \subseteq V(\vec{G})$ such that S is k -connected to T in G if and only if in \vec{G} for each $t \in T'$ the value of a maximum s - t -flow is at least k .

Proof. We consider an embedding of G where all vertices of T are incident to the outer face. Let H denote the directed planar graph obtained by reversing the direction of each arc in G and by adding a new vertex s in the outer face of G connected by arcs sv to all vertices $v \in T$. Further let $T' = S$. Then S is k -connected to T in G if and only if in H there are k internally vertex disjoint s - t -paths for each $t \in T'$ (note that we reversed the direction of all the edges).

We obtain a directed planar graph \vec{G} by splitting each vertex in $V(H) \setminus \{s\}$ into a cycle as follows. Replace each arc e in H , directed from $u \neq s$ to v , by arcs $u_e v_e$ and $v_e v$, where u_e, v_e are new vertices (and distinct for all e). Replace each arc e in H , directed from s to v , by arcs sv_e , and $v_e v$, where v_e is a new vertex. Further for each vertex v in H connect the new vertices v_e by a cycle C_v in the cyclic order of the arcs e around v , where the edges are directed in both directions. Finally a (flow) capacity function is defined where all arcs on cycles C_v receive capacity $1/2$ and all other arcs capacity 1. See Figure 3.

This construction corresponds to the graph $H_{s,t}$ constructed by Kaplan and Nussbaum, except that there is not a specific target t and, additionally the original vertices from H are kept inside of the cycles together with their incoming arcs. In particular \vec{G} is planar and $V(G) \subseteq V(H) \subseteq V(\vec{G})$. Consider some $t \in T' = S$. Note that each s - t -flow in \vec{G} does not use vertices from $V(H) \setminus \{s, t\}$, since these vertices do not have outgoing arcs in \vec{G} . Hence for each $t \in T'$ any s - t -flow in \vec{G} corresponds to an s - t -flow in $H_{s,t}$ (by contracting t and C_t to a single vertex). By Lemma 7 there are k internally vertex disjoint s - t -paths in H if and only if in \vec{G} the value of a maximum s - t -flow is at least k .

Clearly \vec{G} can be constructed in linear time and $|V(\vec{G})| \leq |V(G)| + 2|E(G)| + 1 \leq 7|V(G)|$. This shows that \vec{G} together with s and the set $T' = S$ satisfies the desired conditions. ◀



■ **Figure 4** A CPPT with big angles are marked by solid circles that is not stretchable (a) and the derived directed graph \vec{G} where the highlighted vertices do not have 3 disjoint paths to the outer face (b).

Proof of Theorem 2. First construct a planar directed graph \vec{G} with some $s \in V(\vec{G})$ and $T' \subseteq V(\vec{G})$ with the algorithm from Lemma 8. Then apply an algorithm due to Łącki et al. [24] to \vec{G} , which computes for the source s the maximum flow value to each other vertex in \vec{G} in $O(n \log^3 n)$ time. By Lemma 8 S is k -connected to T in G if and only if in \vec{G} for each $t \in T'$ the value of a maximum s - t -flow is at least k . ◀

4 Recognizing Planar Laman Graphs

Since any Laman graph is 2-connected and 2-connectivity can be checked in linear time, see, e.g., [32], we consider only 2-connected graphs in this section. Further we assume that the graph is given with some plane embedding. As mentioned in the introduction our recognition algorithms are based on the theory due to Haas et al. [16] which they developed in order to show that exactly the planar Laman graphs can be realized as pointed pseudotriangulations. They transfer the concept of PPTs to graphs given with a plane (combinatorial) embedding. A *combinatorial pointed pseudotriangulation* (CPPT) is a plane graph with $2n - 3$ edges and with an assignment of the labels “small”/“big” to the angles satisfying the properties of a PPT. That is, only big angles are incident to the outer face, each inner face contains exactly three small angles and each vertex is incident to exactly one big angle. A CPPT is called *stretchable* if there is a PPT realization (that is, a drawing) of its underlying graph respecting the given labels of the angles and the embedding. Not every CPPT can be stretched to a PPT, see Figure 4(a) for an example. But Haas et al. show that every plane Laman graph admits a CPPT assignment and each CPPT whose underlying graph is Laman is stretchable. This shows that every Laman graph has a realization as a pointed pseudotriangulation. Note that the nonstretchable CPPT in Figure 4(a) contains K_4 as a subgraph which has $6 > 2 \cdot 4 - 3$ edges and hence is not a Laman graph.

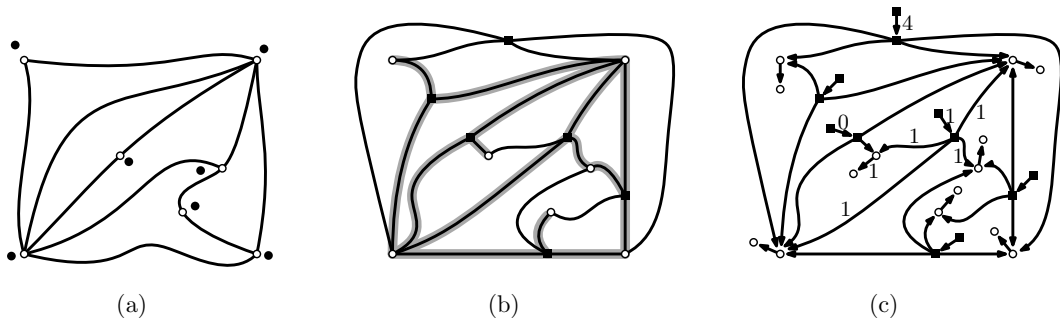
This already outlines how our algorithms check whether a given plane graph G is Laman:

Step 1: Check whether G admits a CPPT assignment and compute one if it exists.

Step 2: Check whether the CPPT from Step 1 is stretchable.

The first step can be reduced to searching for a (generalized) matching in the following planar bipartite graph. The vertex-face incidence graph of a plane graph $G = (V, E)$ with face set F is a planar bipartite graph $H = (V \cup F, E')$ where $(v, f) \in E'$ if and only if $v \in V$

is incident to $f \in F$ in G . Note that for each face each vertex appears at most once along its boundary since we assume that G is 2-connected. A plane graph G admits a CPPT assignment if and only if its vertex-face incidence graph has a subgraph A where each interior face of G has degree 3 in A , the outer face of G has degree 0 in A , and each vertex of G has degree in A equal to its degree in G minus 1. This means that the edges of A correspond to small angles in the CPPT assignment. See Figure 5 for an illustration. Haas et al. provide a simple algorithm with running time $O(n^{3/2})$ checking whether such a subgraph A exists, that is, the algorithm computes a CPPT assignment for a given plane graph if any exists. Note that this algorithm does not compute a drawing and hence does not face the numerical issues mentioned in the introduction.



■ **Figure 5** A plane graph G with a CPPT assignment with big angles marked with a solid circle (a), the vertex-face incident graph of G where vertices corresponding to faces in G are marked with a solid square and the subgraph A corresponding to the CPPT is highlighted in gray (b), the flow network H' constructed from G in the proof of Lemma 9 (some capacities are omitted for better readability) (c).

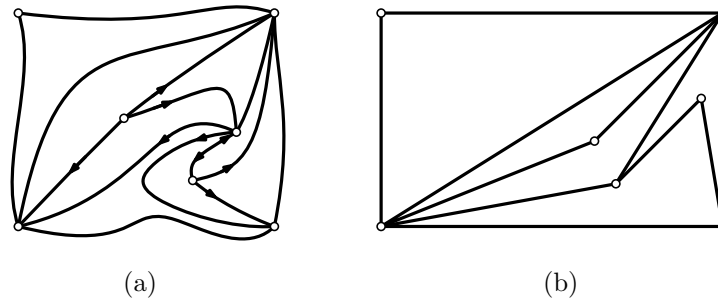
In order to implement the first step with running time in $O(n \log^3 n)$ we use an algorithm by Borradaile et al. [5] computing a maximum flow between multiple sources and sinks.

► **Lemma 9.** *There is an algorithm computing a CPPT assignment for any planar graph G if any exists in $O(n \log^3 n)$ time.*

Proof. Consider the planar directed graph H' whose vertex set consists of two vertices u_1 and u_2 for each $u \in V(G)$ and two vertices f_1, f_2 for each face of G . For each $u \in V(G)$ add an arc $u_1 u_2$ with capacity 1, for each interior face f of G add an arc from $f_1 f_2$ with capacity equal to the number of vertices incident to f minus 3, and for the outer face f of G add an arc $f_1 f_2$ with capacity equal to the number of vertices incident to f . Finally for each vertex u and face f add an arc $f_2 u_1$ with capacity 1 if u is incident to f . Then there is a 1-1-correspondence between CPPT assignments of G and integer flows in H' with sources in $\{f_1 \mid f \text{ face in } G\}$, sinks in $\{u_2 \mid u \in V(G)\}$, and value $|V(G)|$. Note that the size of H' is linear in the size of G and can be constructed in linear time. In particular the algorithm of Borradaile et al. [5] (which computes an integer flow) can be used to compute a maximum flow in $O(n \log^3 n)$ time. If this flow has value $|V(G)|$ we computed a CPPT assignment if any exists, otherwise there is no CPPT assignment. ◀

For the second step, that is, checking whether a given CPPT is stretchable, we use the following characterization of stretchability by Haas et al. Recall that a set $S \subseteq V(G)$ is 3-connected to a set $T \subseteq V(G) \setminus S$ if for each vertex $s \in S$ there are three vertex disjoint directed paths from s to distinct vertices in T . Figures 4 and 6 give illustrations of the following result.

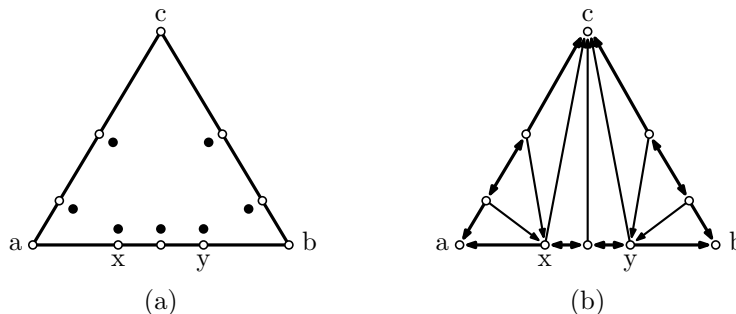
► **Lemma 10** ([16]). *For a CPPT G a directed plane graph \vec{G} , with $V(\vec{G}) = V(G)$, can be computed in linear time such that G is stretchable if and only if the set of interior vertices of \vec{G} is 3-connected to the set of boundary vertices of \vec{G} .*



■ **Figure 6** (a) The derived directed graph \vec{G} from the CPPT in Figure 5(c). (b) The stretched pointed pseudotriangulation.

In fact any directed graph \vec{G} satisfying the following conditions is suitable in the previous lemma (see [16, Lemma 16] and note that G is a CPPT): (i) \vec{G} contains the underlying graph of G and has a planar embedding that respects the embedding of G , (ii) the vertices on the outer face have no outgoing edges, and (iii) every interior vertex v has three outgoing edges where two of them are the extreme edges in G (these are the edges incident to the big angle) and one is an edge to a vertex in the boundary of the face of G containing the big angle at v .

We give a computation of such a graph \vec{G} similar to the one of Hass et al. [16] for completeness. First for each vertex v the two extreme edges incident to its big angle are oriented from v to the respective neighbors (it might happen that some edges are oriented in both directions). Next, we triangulate the underlying graph of G as follows. Consider an inner face of G . If the face contains no big angles, we are done. Otherwise let a , b , and c be the vertices corresponding to the three small angles and let X_{ab} , X_{bc} , and X_{ac} denote the vertices between a and b , between b and c , and between a and c , respectively. Assume that $X_{a,b} \neq \emptyset$. Let x and y denote the vertices in $X_{a,b}$ adjacent to a respectively b (it might happen that $x = y$). Then we add directed diagonals from each vertex in $X_{a,b}$ to c , from each vertex in $X_{a,c}$ to x , and from each vertex in $X_{b,c}$ to y . See Figure 7 for an illustration. We do this for all faces and the obtained triangulation satisfies the conditions (i) – (iii) stated above. Finally we remove all undirected edges. This construction can be implemented in linear time by traversing the boundary of each face once.



■ **Figure 7** (a) An inner face in a CPPT with big angles marked with a solid circle. (b) The directed diagonals added to that face to obtain a directed graph satisfying the assumption of Lemma 10.

Now we are ready to describe how to check whether a given plane graph G is Laman. Note that a graph that admits a CPPT assignment has exactly $2n - e$ edges [16]. The simpler algorithm uses the algorithms of Haas et al. [16] to find a CPPT assignment and the directed plane graph \vec{G} from Lemma 10, and then the algorithm from Theorem 1 to decide whether the set of interior vertices of \vec{G} is 3-connected to the set of boundary vertices. Note that each vertex in \vec{G} is either an inner vertex or on the boundary. This decides whether G is a plane Laman graph by Theorem 1 and Lemma 10 and has running time $O(n^{3/2})$. The faster algorithm works as follows.

Proof of Theorem 3. First, search for a CPPT assignment of G using the algorithm from Lemma 9. If there is no such assignment then G is not Laman. Otherwise G is Laman if and only if the CPPT computed in the first step is stretchable. To check this we compute the auxiliary directed plane graph \vec{G} from Lemma 10 such that the CPPT is stretchable if and only if the set of interior vertices of \vec{G} is 3-connected to the set of boundary vertices of \vec{G} . Note that the boundary vertices of \vec{G} are clearly incident to a common face. Hence we can check the connectivity property using the algorithm from Theorem 2. This algorithm decides whether G is a plane Laman graph by Theorem 2 and Lemma 10 and has running time $O(n \log^3 n)$. ◀

5 Conclusions and further directions

An obvious direction for future research is to search for faster or simpler algorithms recognizing (planar) Laman graphs.

Our algorithms do not provide any certificate for their correctness. This could be a Henneberg sequence [17] or a decomposition into two acyclic subgraphs [7]. We do not know how to compute either of these faster than using the algorithm of Gabow and Westermann [13].

Further our strategy heavily depends on planarity. Thus it seems unlikely that we can extend our approach to nonplanar Laman graphs. Instead it seems interesting to extend our strategies to more general pseudotriangulations. Orden et al. [29] show a characterization of general pseudotriangulations that extends the one for PPTs described in Section 4. Again there is a notion of combinatorial pseudotriangulations (CPT) which are stretchable if and only if certain combinatorial conditions hold. Besides a connectivity condition similar to the one for CPPTs there is an additional condition on sizes of subgraphs. In particular, a CPT might not be stretchable although the underlying graph is realizable as a pseudotriangulation. We do not know how to identify the stretchable CPTs.

Instead of asking if a graph has a representation as a (pointed) pseudotriangulation one can also ask for other representations such as straight-line triangle representations [1]. For this problem no polynomial time algorithm is known, although the problem shares many similarities with the Laman graph recognition problem.

Finally it is of independent interest to see if the connectivity results for directed graphs can be extended. In the last part of the introduction we describe a naive approach checking for a general directed graph whether a set S is k -connected to a set T with time in $O(|S|m)$.

When each vertex is either in S or in T we can adapt the ideas presented in Section 2 for classes of graphs with small separator. Then the running time becomes linear for graphs with separators of constant size and stays in $O(n^{3/2})$ as long as there are separators of size $O(\sqrt{n})$. We do not know how to improve upon the naive approach for general directed graphs when some vertices are not in $S \cup T$.

References

- 1 Nieke Aerts and Stefan Felsner. Straight line triangle representations. *Discrete Comput. Geom.*, 57(2):257–280, 2017.
- 2 Jørgen Bang-Jensen and Gregory Gutin. *Digraphs*. Springer Monographs in Mathematics. Springer, London, second edition, 2009.
- 3 Alex R. Berg and Tibor Jordán. Algorithms for graph rigidity and scene analysis. In *Algorithms—ESA 2003*, volume 2832 of *Lecture Notes in Comput. Sci.*, pages 78–89. Springer, Berlin, 2003.
- 4 Glencora Borradaile, David Eppstein, Amir Nayyeri, and Christian Wulff-Nilsen. All-pairs minimum cuts in near-linear time for surface-embedded graphs. In *32nd International Symposium on Computational Geometry (SoCG'16)*, volume 51 of *LIPICs. Leibniz Int. Proc. Inform.*, pages 22:1–22:16, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.
- 5 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. *SIAM J. Comput.*, 46(4):1280–1303, 2017.
- 6 Robert Connelly, Erik D. Demaine, and Günter Rote. Straightening Polygonal Arcs and Convexifying Polygonal Cycles. *Discrete Comput. Geom.*, 30(2):205–239, 2003.
- 7 Ovidiu Daescu and Anastasia Kurdia. Towards an optimal algorithm for recognizing Laman graphs. *J. Graph Algorithms Appl.*, 13(2):219–232, 2009.
- 8 David Eisenstat and Philip N. Klein. Linear-time algorithms for max flow and multiple-source shortest paths in unit-weight planar graphs. In *45th ACM Symposium on Theory of Computing (STOC'13)*, pages 735–744. ACM, New York, 2013.
- 9 Zsolt Fekete, Tibor Jordán, and Walter Whiteley. An inductive construction for plane Laman graphs via vertex splitting. In *Algorithms—ESA 2004*, volume 3221 of *Lecture Notes in Comput. Sci.*, pages 299–310. Springer, Berlin, 2004.
- 10 Lester R. Ford, Jr. and Delbert R. Fulkerson. *Flows in networks*. Princeton University Press, Princeton, N.J., 1962.
- 11 Harold N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *J. Comput. System Sci.*, 50(2):259–273, 1995.
- 12 Harold N. Gabow. Using expander graphs to find vertex connectivity. *J. ACM*, 53(5):800–844, 2006.
- 13 Harold N. Gabow and Herbert H. Westermann. Forests, frames, and games: algorithms for matroid sums and applications. *Algorithmica*, 7(5-6):465–497, 1992.
- 14 Loukas Georgiadis. Testing 2-vertex connectivity and computing pairs of vertex-disjoint s - t paths in digraphs. In *37th International Colloquium on Automata, Languages and Programming (ICALP'10)*, volume 6198 of *Lecture Notes in Comput. Sci.*, pages 738–749. Springer, Berlin, 2010.
- 15 Loukas Georgiadis, Daniel Graf, Giuseppe F. Italiano, Nikos Parotsidis, and Przemysław Uznański. All-pairs 2-reachability in $O(n^\omega \log n)$ time. In *44th International Colloquium on Automata, Languages, and Programming*, volume 80 of *LIPICs. Leibniz Int. Proc. Inform.*, pages 74:1–74:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.
- 16 Ruth Haas, David Orden, Günter Rote, Francisco Santos, Brigitte Servatius, Herman Servatius, Diane Souvaine, Ileana Streinu, and Walter Whiteley. Planar minimally rigid graphs and pseudo-triangulations. *Comput. Geom.*, 31(1-2):31–61, 2005.
- 17 Lebrecht Henneberg. Die Graphische Statik der Starren Körper. In Felix Klein and Conrad Müller, editors, *Encyklopädie der Mathematischen Wissenschaften mit Einschluss ihrer Anwendungen: Viertes Band: Mechanik*, pages 345–434. Vieweg+Teubner Verlag, Wiesbaden, 1908.
- 18 Jacob Holm, Eva Rotenberg, and Mikkel Thorup. Planar reachability in linear space and constant time. In *56th Annual IEEE Symposium on Foundations of Computer Science (FOCS'15)*, pages 370–389. IEEE Computer Society, Los Alamitos, CA, 2015.

- 19 Giuseppe F. Italiano, Luigi Laura, and Federico Santaroni. Finding strong bridges and strong articulation points in linear time. *Theoret. Comput. Sci.*, 447:74–84, 2012.
- 20 Donald J. Jacobs and Bruce Hendrickson. An algorithm for two-dimensional rigidity percolation: the pebble game. *J. Comput. Phys.*, 137(2):346–365, 1997.
- 21 Tibor Jordán. Combinatorial rigidity: graphs and matroids in the theory of rigid frameworks. In *Discrete geometric analysis*, volume 34 of *MSJ Mem.*, pages 33–112. Math. Soc. Japan, Tokyo, 2016.
- 22 Haim Kaplan and Yahav Nussbaum. Maximum flow in directed planar graphs with vertex capacities. *Algorithmica*, 61(1):174–189, 2011.
- 23 Stephen Kobourov, Torsten Ueckerdt, and Kevin Verbeek. Combinatorial and geometric properties of planar Laman graphs. In *24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'12)*, pages 1668–1678. SIAM, Philadelphia, PA, 2012.
- 24 Jakub Łącki, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Single Source – All Sinks Max Flows in Planar Digraphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS'12)*, pages 599–608. IEEE Computer Society, Washington, DC, 2012.
- 25 Gerard Laman. On graphs and rigidity of plane skeletal structures. *J. Engrg. Math.*, 4:331–340, 1970.
- 26 Audrey Lee and Ileana Streinu. Pebble game algorithms and sparse graphs. *Discrete Math.*, 308(8):1425–1437, 2008.
- 27 Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36(2):177–189, 1979.
- 28 Shay Mozes, Kirill Nikolaev, Yahav Nussbaum, and Oren Weimann. Minimum cut of directed planar graphs in $O(n \log \log n)$ time. In *29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, pages 477–494. SIAM, Philadelphia, PA, 2018.
- 29 David Orden, Francisco Santos, Brigitte Servatius, and Herman Servatius. Combinatorial pseudo-triangulations. *Discrete Math.*, 307(3-5):554–566, 2007.
- 30 James B. Orlin. Max flows in $O(nm)$ time, or better. In *24th ACM Symposium on Theory of Computing (STOC'13)*, pages 765–774. ACM, New York, 2013.
- 31 Günter Rote, Francisco Santos, and Ileana Streinu. Pseudo-triangulations—a survey. In *Surveys on discrete and computational geometry*, volume 453 of *Contemp. Math.*, pages 343–410. Amer. Math. Soc., Providence, RI, 2008.
- 32 Jens M. Schmidt. A simple test on 2-vertex- and 2-edge-connectivity. *Inform. Process. Lett.*, 113(7):241–244, 2013.
- 33 Ileana Streinu. Pseudo-triangulations, rigidity and motion planning. *Discrete Comput. Geom.*, 34(4):587–635, 2005.

Simultaneous Representation of Proper and Unit Interval Graphs

Ignaz Rutter 

Faculty of Computer Science and Mathematics, University of Passau, Germany
rutter@fim.uni-passau.de

Darren Strash 

Department of Computer Science, Hamilton College, Clinton, NY, USA
dstrash@hamilton.edu

Peter Stumpf 

Faculty of Computer Science and Mathematics, University of Passau, Germany
stumpf@fim.uni-passau.de

Michael Vollmer

Department of Informatics, Karlsruhe Institute of Technology (KIT), Germany
michael.vollmer@kit.edu

Abstract

In a confluence of combinatorics and geometry, simultaneous representations provide a way to realize combinatorial objects that share common structure. A standard case in the study of simultaneous representations is the *sunflower case* where all objects share the same common structure. While the recognition problem for general simultaneous interval graphs – the simultaneous version of arguably one of the most well-studied graph classes – is NP-complete, the complexity of the sunflower case for three or more simultaneous interval graphs is currently open. In this work we settle this question for *proper* interval graphs. We give an algorithm to recognize simultaneous proper interval graphs in linear time in the sunflower case where we allow any number of simultaneous graphs. Simultaneous *unit* interval graphs are much more “rigid” and therefore have less freedom in their representation. We show they can be recognized in time $\mathcal{O}(|V| \cdot |E|)$ for any number of simultaneous graphs in the sunflower case where $G = (V, E)$ is the union of the simultaneous graphs. We further show that both recognition problems are in general NP-complete if the number of simultaneous graphs is not fixed. The restriction to the sunflower case is in this sense necessary.

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

Keywords and phrases Intersection Graphs, Recognition Algorithm, Proper/Unit Interval Graphs, Simultaneous Representations

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.80

Related Version A full version of the paper is available at <https://arxiv.org/abs/1908.08882>.

Funding This work was partially supported by grant RU 1903/3-1 of the German Research Foundation (DFG) and by the DFG Research Training Group 2153: “Energy Status Data – Informatics Methods for its Collection, Analysis and Exploitation”.

1 Introduction

Given a family of sets \mathcal{R} , the corresponding *intersection graph* G has a vertex for each set and two vertices are adjacent if and only if their sets have a non-empty intersection. If all sets are intervals on the real line, then \mathcal{R} is an *interval representation* of G and G is an *interval graph*; see Figure 1.

In the context of intersection graph classes, much work has been devoted to efficiently computing a *representation*, which is a collection of sets or geometric objects having an intersection graph that is isomorphic to a given graph. For many well-known graph classes, such as interval graphs and chordal graphs, this is a straightforward task [14, 28]. However,



© Ignaz Rutter, Darren Strash, Peter Stumpf, and Michael Vollmer;
licensed under Creative Commons License CC-BY

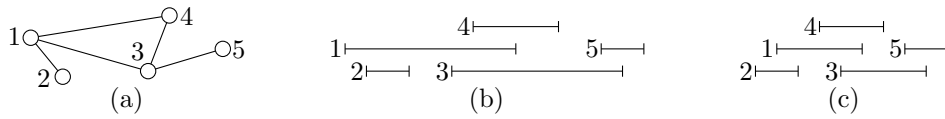
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 80; pp. 80:1–80:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** (a) A graph, with (b) an interval representation and (c) proper interval representation.

often it is desirable to consistently represent *multiple* graphs that have subgraphs in common. This is true, for instance, in realizing schedules with shared events, embedding circuit graphs of adjacent layers on a computer chip, and visualizing the temporal relationship of graphs that share a common subgraph [19]. Likewise, in genome reconstruction, we can ask if a sequence of DNA can be reconstructed from strands that have sequences in common [13].

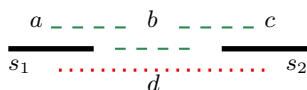
Simultaneous representations capture this in a very natural way. Given *simultaneous graphs* G_1, G_2, \dots, G_k where each pair of graphs G_i, G_j share some common subgraph, a *simultaneous representation* asks for a fixed representation of each vertex that gives a valid representation of each G_i . This notion is closely related to *partial representation extension*, which asks if a given (fixed) representation of a subgraph can be extended to a representation of the full graph. Partial representation extension has been extensively studied for graph classes such as interval graphs [20], circle graphs [8], as well as proper and unit interval graphs [20]. For interval graphs, Bläsius and Rutter [3] have even shown that the partial interval representation problem can be reduced to a simultaneous interval representation problem on two graphs in linear time.

Simultaneous representations were first studied in the context of embedding graphs [2, 7], where the goal is to embed each simultaneous graph without edge crossings while shared subgraphs have the same induced embedding. Unsurprisingly, many variants are NP-complete [12, 26, 1, 11]. The notion of simultaneous representation of general intersection graph classes was introduced by Jampani and Lubiw [19], who showed that it is possible to recognize simultaneous chordal graphs with two graphs in polynomial time, and further gave a polynomial time algorithm to recognize simultaneous comparability graphs and permutation graphs with two or more graphs that share the same subgraph (the *sunflower case*). They further showed that recognizing three or more simultaneous chordal graphs is NP-complete.

Golumbic et al. [15] introduced the *graph sandwich problem* for a graph class Π . Given a vertex set V and edge sets $E_1 \subseteq E_2 \subseteq \binom{V}{2}$ it asks whether there is an edge set $E_1 \subseteq E \subseteq E_2$ such that the *sandwich graph* $G = (V, E)$ is in Π . Jampani and Lubiw showed that if Π is an intersection graph class, then recognizing k simultaneous graphs in Π in the sunflower case is a special case of the graph sandwich problem where $(V, E_2 \setminus E_1)$ is a k -partite graph [19].

We consider simultaneous *proper* and *unit* interval graphs. An interval graph is proper if in an interval representation no interval properly contains another one (see Figure 1), and it is unit if all intervals have length one. Interestingly, while proper and unit interval graphs are the same graph class as shown by Roberts [25], simultaneous unit interval graphs differ from simultaneous proper interval graphs; see Figure 2. Unit interval graphs are intersection graphs and therefore the graph sandwich paradigm described by Jampani and Lubiw applies. Proper interval graphs are not since in a simultaneous representation intervals of distinct graphs may contain each other which means that the intersection graph of all intervals in the simultaneous representation is not proper.

Sunflower (*unit*) interval graphs are a generalization of *probe* (proper) interval graphs, where each sunflower graph has only one non-shared vertex. Both variants of probe graphs can be recognized in linear time [22, 23].



■ **Figure 2** A simultaneous proper interval representation of a sunflower graph \mathcal{G} consisting of two paths $G_1 = (s_1, a, b, c, s_2)$ (dashed) and $G_2 = (s_1, d, s_2)$ (dotted) with shared start and end s_1, s_2 (bold). They have no simultaneous unit interval representation: The intervals a and c enforce that b lies between s_1 and s_2 . Interval d therefore includes b in every simultaneous proper interval representation. In particular, not both can have size one.

Simultaneous interval graphs were first studied by Jampani and Lubiw [18] who gave a $\mathcal{O}(n^2 \lg n)$ -time recognition algorithm for the special case of two simultaneous graphs. Bläsius and Rutter [3] later showed how to recognize two simultaneous interval graphs in linear time. Bok and Jedličková showed that the recognition of an arbitrary number of simultaneous interval graphs is in general NP-complete [4]. However, the complexity for the sunflower case with more than two simultaneous graphs is still open.

Our Results. We settle these problems with k not fixed for simultaneous *proper* and *unit* interval graphs – those graphs with an interval representation where no interval properly contains another and where all intervals have unit length, respectively [10, 27, 9, 16]. For the sunflower case, we provide efficient recognition algorithms. The running time for proper interval graphs is linear, while for the unit case it is $\mathcal{O}(|V| \cdot |E|)$ where $G = (V, E)$ is the union of the sunflower graphs. In the full version we prove NP-completeness for the non-sunflower case. The reductions are similar to the simultaneous independent work of Bok and Jedličková for simultaneous interval graphs [4].

Organization. We begin by introducing basic notation and existing tools throughout Section 2. In Section 3 we give a characterization of simultaneous proper interval graphs, from which we develop an efficient recognition algorithm. In Section 4 we characterize simultaneous proper interval graphs that can be simultaneous unit interval graphs, and then exploit this property to efficiently search for a representation among simultaneous proper interval graph representations. Proofs of lemmas and theorems marked with \star are omitted.

2 Preliminaries

In this section we give basic notation, definitions and characterizations. Section 2.1 collects basic concepts on graph theory, orderings, and PQ-trees. Section 2.2 introduces (proper) interval graphs and presents relations between the representations of such graphs and their induced subgraphs. Finally, Section 2.3 introduces the definition and notation of simultaneous graphs.

2.1 Graphs, Orderings, and PQ-trees

Unless mentioned explicitly, all graphs in this paper are undirected. For a graph $G = (V, E)$ we denote its size $|G| := |V| + |E|$.

Let σ be a binary relation. Then we write $a_1 \leq_\sigma a_2$ for $(a_1, a_2) \in \sigma$, and we write $a_1 <_\sigma a_2$ if $a_1 \leq_\sigma a_2$ and $a_1 \neq a_2$. We omit the subscript and simply use $<$ and \leq if the ordering it refers to is clear from the context. We denote the *reversal* of a linear order σ by σ^r , and we use \circ to concatenate linear orders of disjoint sets.

A *PQ-tree* is a data structure for representing sets of linear orderings of a ground set X . Namely, given a set $\mathcal{C} \subseteq 2^X$, a *PQ-tree on X for \mathcal{C}* is a tree data structure T that represents the set $\text{CONSISTENT}(T)$ containing exactly the linear orders of X in which the elements of each set $C \in \mathcal{C}$ are consecutive. The PQ-tree T can be computed in time $O(|X| + \sum_{C \in \mathcal{C}} |C|)$ [6]. Given a PQ-tree T on the set X and a subset $X' \subseteq X$, there exists a PQ-tree T' , called the *projection* of T to X' , that represents exactly the linear orders of X' that are restrictions of orderings in $\text{CONSISTENT}(T)$. For any two PQ-trees T_1 and T_2 on the set X , there exists a PQ-tree T with $\text{CONSISTENT}(T) = \text{CONSISTENT}(T_1) \cap \text{CONSISTENT}(T_2)$, called the *intersection* of T_1 and T_2 . Both the projection and the intersection can be computed in $O(|X|)$ time [5].

2.2 Interval Graphs, Proper Interval Graphs, and Their Subgraphs

An *interval representation* $R = \{I_v \mid v \in V\}$ of a graph $G = (V, E)$ associates with each vertex $v \in V$ an interval $I_v = [x, y]$ of real numbers such that for each pair of vertices $u, v \in V$ we have $I_u \cap I_v \neq \emptyset$ if and only if $\{u, v\} \in E$, i.e., the intervals intersect if and only if the corresponding vertices are adjacent. An interval representation R is *proper* if no interval properly contains another one, and it is *unit* if all intervals have length 1. A graph is an *interval graph* if and only if it admits an interval representation, and it is a *proper (unit) interval graph* if and only if it admits a proper (unit) interval representation. It is well-known that proper and unit interval graphs are the same graph class.

► **Proposition 1** ([25]). *A graph is a unit interval graph if and only if it is a proper interval graph.*

However, this does not hold in the simultaneous case where every simultaneous unit interval representation is clearly a simultaneous proper interval representation of the same graph, but not every simultaneous proper interval representation implies a simultaneous unit interval representation; see Figure 2.

We use the well-known characterization of proper interval graphs using *straight enumerations* [10]. Two adjacent vertices $u, v \in V$ are *indistinguishable* if we have $N[u] = N[v]$ where $N[u] = \{v : uv \in E(H)\} \cup \{u\}$ is the closed neighborhood. Being indistinguishable is an equivalence relation and we call the equivalence classes *blocks* of G . We denote the block of G that contains vertex u by $B(u, G)$. Note that for a subgraph $G' \subseteq G$ the block $B(u, G')$ may contain vertices in $V(G') \setminus B(u, G)$ that have the same neighborhood as u in G' but different neighbors in G . Two blocks B, B' are *adjacent* if and only if $uv \in E$ for (any) $u \in B$ and $v \in B'$. A linear order σ of the blocks of G is a *straight enumeration* of G if for every block, the block and its adjacent blocks are consecutive in σ . A proper interval representation R defines a straight enumeration $\sigma(R)$ by ordering the intervals by their starting points and grouping together the blocks. Conversely, for each straight enumeration σ , there exists a corresponding representation R with $\sigma = \sigma(R)$ [10]. A *fine enumeration* of a graph H is a linear order η of $V(H)$ such that for $u \in V(H)$ the closed neighborhood $N_H[u]$ is consecutive in η .

► **Proposition 2** ([24, 10, 17]). *(i) A graph is a proper interval graph if and only if it has a fine enumeration. (ii) A graph is a proper interval graph if and only if it admits a straight enumeration. (iii) A straight enumeration of a connected proper interval graph is unique up to reversal.*

2.3 Simultaneous Graphs

A *simultaneous graph* is a tuple $\mathcal{G} = (G_1, \dots, G_k)$ of graphs G_i that may each share vertices and edges. Note that this definition differs from the one we gave in the introduction. This way the input for the simultaneous representation problem is a single entity. The size $|\mathcal{G}|$ of a simultaneous graph is $\sum_{i=1}^k |G_i|$. We call \mathcal{G} *connected*, if $\bigcup_{i=1}^k G_i$ is connected. A *simultaneous (proper/unit) interval representation* $\mathcal{R} = (R_1, \dots, R_k)$ of \mathcal{G} is a tuple of representations such that $R_i \in \mathcal{R}$ is a (proper/unit) interval representation of graph G_i and the intervals representing shared vertices are identical in each representation. A simultaneous graph is a *simultaneous (proper/unit) interval graph* if it admits a simultaneous (proper/unit) interval representation.

An important special case is that of *sunflower graphs*. The simultaneous graph \mathcal{G} is a *sunflower graph* if each pair of graphs G_i, G_j with $i \neq j$ shares exactly the same subgraph S , which we then call the *shared graph*. Note that, for \mathcal{G} to be a simultaneous interval graph, it is a necessary condition that $G_i \cap G_j$ is an induced subgraph of G_i and G_j for $i, j = 1, \dots, k$. In particular, in the sunflower case the shared graph S must be an induced subgraph of each G_i . The following lemma allows us to restrict ourselves to instances whose union graph $\bigcup_{\mathcal{G}} = \bigcup_{i=1}^k G_i$ is connected.

► **Lemma 3** (\star). *Let $\mathcal{G} = (G_1, \dots, G_k)$ be a simultaneous graph and let C_1, \dots, C_l be the connected components of $\bigcup_{\mathcal{G}}$. Then \mathcal{G} is a simultaneous (proper) interval graph if and only if each of the graphs $\mathcal{G}_i = (G_1 \cap C_i, \dots, G_k \cap C_i)$, $i = 1, \dots, l$ is a simultaneous (proper/unit) interval graph.*

3 Sunflower Proper Interval Graphs

In this section, we deal with simultaneous proper interval representations of sunflower graphs. We first present a combinatorial characterization of the simultaneous graphs that admit such a representation. Afterwards, we present a simple linear-time recognition algorithm. Finally, we derive a combinatorial description of all the combinatorially different simultaneous proper interval representations of a connected simultaneous graph, which is a prerequisite for the unit case.

3.1 Characterization

Let $G = (V, E)$ be a proper interval graph with straight enumeration σ and let $V_S \subseteq V$ be a subset of vertices. We call σ *compatible* with a linear order ζ of V_S if, we have for $u, v \in V_S$ that $u \leq_{\zeta} v$ implies $B(u, G) \leq_{\sigma} B(v, G)$.

► **Lemma 4.** *Let $\mathcal{G} = (G_1, \dots, G_k)$ be a sunflower graph with shared graph $S = (V_S, E_S)$. Then \mathcal{G} admits a simultaneous proper interval representation \mathcal{R} if and only if there exists a linear order ζ of V_S and straight enumerations σ_i for each G_i that are compatible with ζ .*

Proof Sketch. For a given representation \mathcal{R} the straight enumerations $\sigma_i = \sigma(R_i)$ and linear order ζ of V_S given by their left endpoints in \mathcal{R} clearly satisfy the lemma. Conversely we build a linear order of interval endpoints from each σ_i that equals a proper interval representation. As each σ_i is compatible with ζ , all endpoint orderings allow the same ordering for vertices in S , thus permitting one global ordering of all endpoints. Drawing the intervals according to this ordering then yields a simultaneous representation \mathcal{R} since it extends each individual ordering. ◀

Let $\mathcal{G} = (G_1, \dots, G_k)$ be a sunflower graph with shared graph $S = (V_S, E_S)$ and for each $G_i \in \mathcal{G}$ let σ_i be a straight enumeration of G_i . We call the tuple $(\sigma_1, \dots, \sigma_k)$ a *simultaneous enumeration* if for any $i, j \in \{1, \dots, k\}$ and $u, v \in V_S$ we have $B(u, G_i) <_{\sigma_i} B(v, G_i) \Rightarrow B(u, G_j) \leq_{\sigma_j} B(v, G_j)$. That is, the blocks containing vertices of the shared graph are not ordered differently in any straight enumeration.

► **Theorem 5** (\star). *Let $\mathcal{G} = (G_1, \dots, G_k)$ be a sunflower graph. There exists a simultaneous proper interval representation $\mathcal{R} = (R_1, \dots, R_k)$ of \mathcal{G} if and only if there is a simultaneous enumeration $(\sigma_1, \dots, \sigma_k)$ of \mathcal{G} . If $(\sigma_1, \dots, \sigma_k)$ exists, there also exists \mathcal{R} with $\sigma(R_i) = \sigma_i$ for each $R_i \in \mathcal{R}$.*

3.2 A Simple Recognition Algorithm

In this section we develop a very simple recognition algorithm for sunflower graphs that admits a simultaneous proper interval representation based on Theorem 5.

Let $\mathcal{G} = (G_1, \dots, G_k)$ be a sunflower graph with shared graph $S = (V_S, E_S)$. By Proposition 2, for each graph G_i , there exists a PQ-tree T'_i that describes exactly the fine enumerations of G_i . We denote by $T_i = T'_i|S$ the projection of T'_i to the vertices in S . The tree T_i thus describes all proper interval representations of S that can be extended to a proper interval representation of G_i . Let T denote the intersection of T_1, \dots, T_k . By definition, T represents all proper interval representations of S that can be extended to a proper interval representation of each graph G_i . Thus, \mathcal{G} admits a simultaneous proper interval representation if and only if T is not the null-tree.

If T is not the null-tree, we can obtain a simultaneous representation by choosing any ordering $O \in \text{CONSISTENT}(T)$ and constructing a simultaneous representation \mathcal{S} of S . Using the algorithm of Klavík et al. [20], we can then independently extend \mathcal{S} to representations R_i of G_i . Since the trees T_i can be computed in time linear in the size of the graph G_i , and the intersection of two trees takes linear time, the testing algorithm takes time linear in the total size of the G_i . The representation extension of Klavík et al. [20] runs in linear time. We therefore have the following theorem.

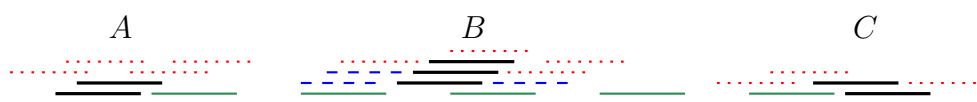
► **Theorem 6.** *Given a sunflower graph $\mathcal{G} = (G_1, \dots, G_k)$, it can be tested in linear time whether \mathcal{G} admits a simultaneous proper interval representation.*

3.3 Combinatorial Description of Simultaneous Representations

Let \mathcal{G} be a sunflower proper interval graph with shared graph S and simultaneous representation \mathcal{R} . Then, each representation $R \in \mathcal{R}$ uses the same intervals for vertices of S and implies the same straight enumeration $\sigma_S(\mathcal{R}) = \sigma_S(R) = \sigma(\{I_v \in R : v \in V(S)\})$.

► **Lemma 7.** *Let \mathcal{G} be a connected sunflower proper interval graph with shared graph S . Across all simultaneous proper interval representations \mathcal{R}' of \mathcal{G} , the straight enumeration $\sigma_S(\mathcal{R}')$ of S is unique up to reversal.*

Proof. Let \mathcal{R} be a simultaneous representation of \mathcal{G} and $\sigma_S(\mathcal{R})$ the straight enumeration of S induced by \mathcal{R} . Since \mathcal{G} is connected, for any two blocks B_i and B_{i+1} of S consecutive in $\sigma_S(\mathcal{R})$, there exists a graph $G \in \mathcal{G}$ such that B_i and B_{i+1} are in the same connected component of G . Since S is an induced subgraph of G , for any two vertices $u, v \in V(S)$ with $B(u, S) \neq B(v, S)$ we have $B(u, G) \neq B(v, G)$. This means that a straight enumeration of G implies a straight enumeration of S . Additionally, the straight enumeration of each connected component of G is unique up to reversal by Proposition 2. As a result, for any



■ **Figure 3** Simultaneous proper interval representation of G_1 (green solid), G_2 (red dotted), G_3 (blue dashed) with shared graph S (black bold). S has three blocks A , B , C . We denote the component of G_i containing a block D by C_D^i . C_A^2 , C_B^2 , C_B^3 , C_C^2 are loose. C_A^2 is independent. (C_B^2, C_B^3) is a reversible part. (C_C^2) is not a reversible part, since C_C^1 is aligned at C and not loose.

proper interval representation R of G , the blocks B_i and B_{i+1} are consecutive in $\sigma_S(R)$. This holds for any two consecutive blocks in σ , which means that the consecutivity of all blocks of S is fixed for all simultaneous representations of \mathcal{G} . As a consequence $\sigma_S(\mathcal{R})$ is fixed up to complete reversal. ◀

Let G be a proper interval graph consisting of the connected components C_1, \dots, C_k with straight enumerations $\sigma_1, \dots, \sigma_k$. Let $\sigma_1 \circ \dots \circ \sigma_k$ be a straight enumeration of G . Then we say the straight enumeration $\sigma' = \sigma_1 \circ \dots \circ \sigma_{i-1} \circ \sigma_i^r \circ \sigma_{i+1} \circ \dots \circ \sigma_k$ is *obtained from σ by reversal of C_i* . For a sunflower graph \mathcal{G} containing G with shared graph $S = (V_S, E_S)$, we call a component $C = (V_C, E_C)$ of G *loose*, if all vertices $V_S \cap V_C$ are in the same block of S . Reversal of loose components is the only “degree of freedom” among simultaneous enumerations, besides full reversal, and is formally shown in the full version.

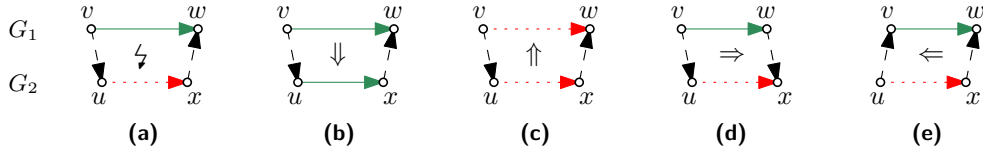
To obtain a complete characterization, we now introduce additional terms to specify which reversals result in simultaneous enumerations (see Figure 3). Let $\mathcal{G} = (G_1, \dots, G_k)$ be a connected sunflower proper interval graph with shared graph S . We say a component C of a graph in \mathcal{G} *aligns* two vertices $u, v \in S$ if they are in different blocks of C , i.e., $B(u, C) \neq B(v, C)$. If in addition u and v are in the same block B of S , we say C is *oriented at B* . If there is another component C' among graphs in \mathcal{G} oriented at B , the orientation of their straight enumerations in a simultaneous enumeration of \mathcal{G} are dependent; that is, they cannot be reversed independently. This is shown formally in the full version.

For each block B of S , let $\mathcal{C}(B)$ be the connected components among graphs in \mathcal{G} oriented at B . Since a component may contain B without aligning vertices, we have $0 \leq |\mathcal{C}(B)| \leq k$. If $\mathcal{C}(B)$ contains only loose components, we call it a *reversible part*. Note that a reversible part $\mathcal{C}(B)$ contains at most one component of each graph G_i . Additionally, we call a loose component C *independent*, if it does not align any two vertices of S . Let $(\sigma_1, \dots, \sigma_k)$ and $(\sigma'_1, \dots, \sigma'_k)$ be tuples of straight enumerations of G_1, \dots, G_k . We say $(\sigma'_1, \dots, \sigma'_k)$ is *obtained from $(\sigma_1, \dots, \sigma_k)$ through reversal of reversible part $\mathcal{C}(B)$* , if $\sigma'_1, \dots, \sigma'_k$ are obtained by reversal of all components in $\mathcal{C}(B)$.

► **Theorem 8** (\star). *Let $\mathcal{G} = (G_1, \dots, G_k)$ be a connected sunflower graph with shared graph S and simultaneous enumeration $\rho = (\sigma_1, \dots, \sigma_k)$. Then $\rho' = (\sigma'_1, \dots, \sigma'_k)$ is a simultaneous enumeration of \mathcal{G} if and only if ρ' can be obtained from ρ or ρ^r through reversal of independent components and reversible parts.*

4 Sunflower Unit Interval Graphs

In the previous section we characterized all simultaneous enumerations for a sunflower proper interval graph \mathcal{G} . We say a simultaneous proper/unit interval representation of a sunflower graph \mathcal{G} *realizes* a simultaneous enumeration $\zeta = (\zeta_1, \dots, \zeta_k)$ of ζ , if for $i \in \{1, \dots, k\}$ the representation of G_i corresponds to the straight enumeration ζ_i . In Section 4.1 we provide a criterion which determines for a given simultaneous enumeration ζ of \mathcal{G} whether there is a



■ **Figure 4** (a): The forbidden configuration of Corollary 11. (b)–(e): The four implications of Corollary 12.

simultaneous unit interval representation of \mathcal{G} that realizes ζ . Namely, the criterion is the avoidance of a certain configuration in a partial vertex order of $\bigcup_{\mathcal{G}}$ induced by ζ . In Section 4.2 we combine these findings to efficiently recognize simultaneous unit interval graphs.

4.1 Simultaneous Enumerations of Sunflower Unit Interval Graphs

We first obtain a combinatorial characterization by reformulating the problem of finding a representation as a restricted graph sandwich problem [15].

► **Lemma 9** (\star). *A sunflower graph \mathcal{G} has a simultaneous unit interval representation that realizes a simultaneous enumeration $\zeta = (\zeta_1, \dots, \zeta_k)$ if and only if there is some graph H with $V(H) = V(\mathcal{G})$ that contains the graphs G_1, \dots, G_k as induced subgraphs and has a fine enumeration σ such that for $i \in \{1, \dots, k\}$ straight enumeration ζ_i is compatible with σ on V_i .*

Our approach is to obtain more information on what graph H and the fine enumeration σ must look like. We adapt a characterization of Looges and Olariu [21] to obtain four implications that can be used given only partial information on H and σ (as given by Lemma 9); see Figure 4. For the figures in this section we use arrows to represent a partial order between two vertices. We draw them solid green if they are adjacent, red dotted if they are non-adjacent in some graph G_i , and black dashed if they may or may not be adjacent.

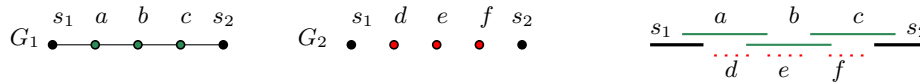
► **Theorem 10** (Looges and Olariu [21]). *A vertex order of a graph $H = (V, E)$ is a fine enumeration if and only if for $v, u, w \in V$ with $v <_{\sigma} u <_{\sigma} w$ and $vw \in E$ we have $vu, uw \in E$.*

► **Corollary 11** (\star). *A vertex order of a graph $H = (V, E)$ is a fine enumeration if and only if there are no four vertices $v, u, x, w \in V$ with $v \leq_{\sigma} u \leq_{\sigma} x \leq_{\sigma} w$ and $vw \in E$ and $ux \notin E$.*

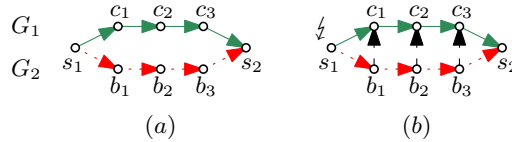
► **Corollary 12**. *Let $H = (V, E)$ be a graph with fine enumeration σ . Let $v, u, x, w \in V$ and $u \leq_{\sigma} x$ as well as $v \leq_{\sigma} w$. Then we have (see Figure 4):*

- (i) $vw \in E \wedge v \leq_{\sigma} u \wedge x \leq_{\sigma} w \Rightarrow ux \in E$
- (ii) $ux \notin E \wedge v \leq_{\sigma} u \wedge x \leq_{\sigma} w \Rightarrow vw \notin E$
- (iii) $vw \in E \wedge ux \notin E \wedge v \leq_{\sigma} u \Rightarrow w <_{\sigma} x$
- (iv) $vw \in E \wedge ux \notin E \wedge x \leq_{\sigma} w \Rightarrow u <_{\sigma} v$.

Now we introduce the forbidden configurations for *simultaneous* enumerations of sunflower unit interval graphs. Throughout this section let $\mathcal{G} = (G_1, \dots, G_k)$ be a sunflower graph with shared graph S and simultaneous enumeration $\zeta = (\zeta_1, \dots, \zeta_k)$. Furthermore, let $V_i = V(G_i)$ and $E_i = E(G_i)$, for $i \in \{1, \dots, k\}$. Finally, let $V = V_1 \cup \dots \cup V_k$. For a straight enumeration η of some graph H we say for $u, v \in V(H)$ that $u <_{\eta} v$, if u is in a block before v , and we say $u \leq_{\eta} v$, if $u = v$ or $u <_{\eta} v$. We call \leq_{η} the *partial order on $V(H)$ corresponding to η* . Note that for distinct u, v in the same block we have neither $u >_{\eta} v$ nor $u \leq_{\eta} v$. We write $u \leq_i v$ and $u <_i v$ instead of $u \leq_{\zeta_i} v$ and $u <_{\zeta_i} v$, respectively.



■ **Figure 5** A sunflower graph $\mathcal{G} = (G_1, G_2)$ with shared vertices s_1, s_2 (black, bold). Let ζ be the simultaneous enumeration realized by the given simultaneous proper interval representation. In (G_1, ζ_1) we have the (s_1, s_2) -chain $C = (s_1, a, b, c, s_2)$ of size 5 (green, solid). In (G_2, ζ_2) we have the (s_1, s_2) -bar $B = (s_1, d, e, f, s_2)$ of size 5 (red, dotted). Hence, sunflower graph \mathcal{G} has the conflict (C, B) for the simultaneous enumeration ζ .



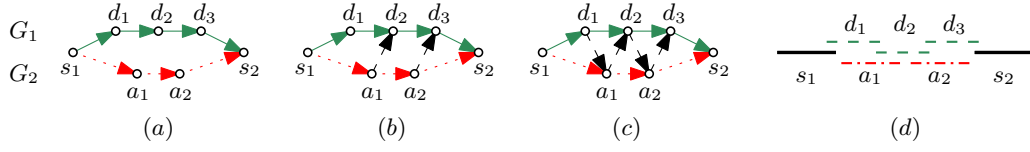
■ **Figure 6** (a): A simultaneous enumeration with conflict. (b): Result with added orderings after scouting, starting at s_2 and finding the conflict in s_1 .

Let $u, v \in V(S)$ with $u \neq v$. For $i \in \{1, \dots, k\}$ a (u, v) -chain of size $m \in \mathbb{N}$ in (G_i, ζ_i) is a sequence $(u = c_1, \dots, c_m = v)$ of vertices in V_i with $c_1 <_i \dots <_i c_m$ that corresponds to a path in G_i . A (u, v) -bar between u and v of size $m \in \mathbb{N}$ in (G_i, ζ_i) is a sequence $(u = b_1, \dots, b_m = v)$ of vertices in V_i with $b_1 <_i \dots <_i b_m$ that corresponds to an independent set in G_i . An example is shown in Figure 5. If there is a (u, v) -chain C in G_i of size $l \geq 2$ and a (u, v) -bar B in (G_j, ζ_j) of size at least l , then we say that (C, B) is a (u, v) -*(chain-bar)-conflict* and that \mathcal{G} has conflict (C, B) for ζ . Note that one can reduce the size of a larger (u, v) -bar by removing intervals between u and v . Thus, we can always assume that in a conflict, we have a bar and a chain of the same size $l \geq 2$. Assume \mathcal{G} has a simultaneous unit interval representation realizing ζ . If a graph $G \in \mathcal{G}$ has a (u, v) -chain of size $l \geq 2$, then the distance between the intervals I_u, I_v for u, v is smaller than $l - 2$. On the other hand, if a graph $G \in \mathcal{G}$ has a (u, v) -bar of size l , then the distance between I_u, I_v is greater than $l - 2$. Hence, sunflower graph \mathcal{G} has no conflict. The result of this section is that the absence of conflicts is not only necessary, but also sufficient.

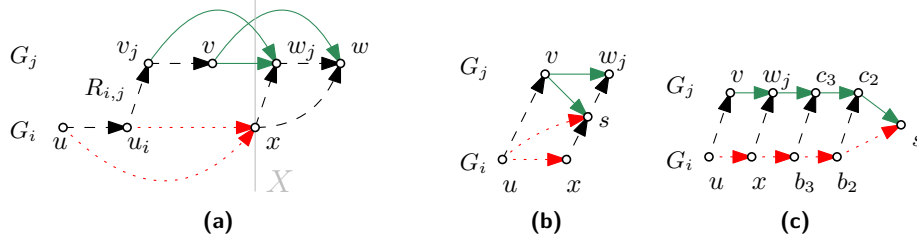
► **Theorem 13.** *Let \mathcal{G} be a sunflower proper interval graph with simultaneous enumeration ζ . Then \mathcal{G} has a simultaneous unit interval representation that realizes ζ if and only if \mathcal{G} has no conflict for ζ .*

Recall that $V_i = V(G_i)$ for $i \in \{1, \dots, k\}$ and $V = V_1 \cup \dots \cup V_k$. Let α^* be the union of the partial orders on V_1, \dots, V_k corresponding to ζ_1, \dots, ζ_k . Set α to be the transitive closure of α^* . We call α the *partial order on V induced by ζ* . The rough idea is that the partial order on V induced by the simultaneous enumeration ζ is extended in two sweeps to a fine enumeration of some graph H that contains G_1, \dots, G_k as induced subgraphs; see Figures 6, 7. For $(u, v) \in \alpha$ we consider u to be to the left of v . The first sweep (*scouting*) goes from the right to the left and makes only necessary extensions according to Corollary 12 (iv). If there is a conflict, then it is found in this step. Otherwise, we can greedily order the vertices on the way back by additionally respecting Corollary 12 (iii) (*zipping*) to obtain a linear extension where both implications are satisfied. In the last step we decide which edges H has by respecting Corollary 12 (i).

For $h \in \{1, \dots, k\}$ we say two vertices $u, v \in V_h$ are *indistinguishable in \mathcal{G}* if we have $N_{G_i}(u) = N_{G_i}(v)$ for all $i \in \{1, \dots, k\}$ with $u, v \in V_i$. In that case u, v can be represented by the same interval in any simultaneous proper interval representation. Thus, we identify



■ **Figure 7** (a): A simultaneous enumeration without conflict. (b): Result with added orderings after scouting. (c): Resulting linear order after zipping. Note that a_1 comes before d_2 in the linear order thanks to scouting. Choosing otherwise would imply a contradiction at s_2 . (d): Resulting unit interval representation for the sandwich graph.



■ **Figure 8** (a): The vertices u_i, v_j, w_j as derived from x and X . The introduced ordering (u_i, v_j) is marked with $R_{i,j}$. (b),(c): Both cases of a chain-bar pair for u and v .

indistinguishable vertices. If $u, v \in V_h$ are not indistinguishable, then we have $N_{G_j}(u) \neq N_{G_j}(v)$ for some $j \in \{1, \dots, k\}$. In that case u, v are ordered by ζ_j and therefore by α . That is, we can assume α to be a linear order on V_i for $i \in \{1, \dots, k\}$. Note that u, v may be ordered even if they are indistinguishable in some input graphs.

For $i \in \{1, \dots, k\}$, let $G_i^c = (V_i, (V_i^c) \setminus E_i)$ be the complement of G_i . We set $E = \{(u, v) \in \alpha \mid uv \in E_1 \cup \dots \cup E_k\}$ and $F = \{(u, v) \in \alpha \mid uv \in E(G_1^c \cup \dots \cup G_k^c)\}$. We call a partial order σ on V *left-closed* if we have

$$\forall v, w, u, x \in V: (vw \in E \wedge ux \in F \wedge x \leq_\sigma w) \Rightarrow u <_\sigma v. \quad (1)$$

Note that a fine enumeration of a graph H with G_1, \dots, G_k as induced subgraphs is left-closed by Corollary 12 (iv). We describe the result of the first sweep with the following lemma.

► **Lemma 14.** *A sunflower graph \mathcal{G} has no conflict for a simultaneous enumeration ζ if and only if there is a left-closed partial order τ that extends the partial order on $V(\mathcal{G})$ induced by ζ .*

Proof Sketch. If there is a conflict (C, B) , then the partial order α induced by ζ cannot be extended to be left-closed since then for $i \in \{1, \dots, k-1\}$ the i 'th vertex of C and B must be ordered and distinct while the first vertex is shared; see Figure 6.

Otherwise, we process the vertices from the right to the left and add for each of them the implied orderings (each is considered as vertex x in the definition of left-closed). First consider the case of just two input graphs G_1, G_2 . Let X be the set of already processed vertices and let σ be the current partial order. We next process a maximal vertex $x \in V \setminus X$. Let $x \in V_i$. Then we choose u_i to be the rightmost vertex in V_i with $u_i x \in F$ and for $j \neq i$ we choose w_j to be the leftmost vertex in V_j with $x \leq w_j$ and v_j to be the leftmost vertex in V_j with $v_j w_j \in E$; see Figure 8a. Each of u_i, v_j, w_j may not exist. If they do, we extend σ to σ' by adding the ordering $u_i \leq_{\sigma'} v_j$. The other implied orderings are exactly those obtained by transitive closure.

Two vertices $u \in V_1, v \in V_2$ are only ordered by α if there is a shared vertex s with $u \leq_\alpha s \leq_\alpha v$ or $v \leq_\alpha s \leq_\alpha u$. The key observation is that if u is ordered before v due to a necessary extension, then there is a shared vertex s and a (v, s) -chain and a (u, s) -bar of equal

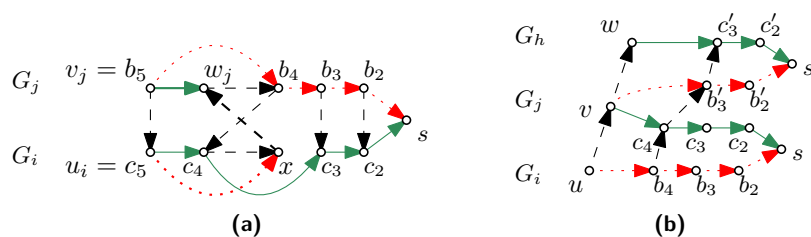


Figure 9 (a): Example situation for $(v_j, u_i) \in \tau_{i,j} \setminus \alpha^*$. We have the (v_j, s) -bar (v_j, b_4, b_3, b_2, s) and the (u_i, s) -chain (u_i, c_4, c_3, c_2, s) and obtain $x \leq_{\tau_{i,j}} w_j \leq_{\tau_{i,j}} b_4 \leq_{\tau_{i,j}} c_4 \leq_{\tau_{i,j}} x$. (b): Example situation for the transitivity of τ where we have a chain-bar pair for u, v as well as for v, w . We obtain $b_4 \leq_{\tau_{i,j}} c_4 \leq_{\tau_{i,j}} b'_3 \leq_{\tau_{i,j}} c'_3$ and since $u <_{\alpha} b_4$ and $w <_{\alpha} c'_3$ we get $b_4 \leq_{\tau_{i,h}} c'_3$ in an appropriate induction and with $\tau_{i,h}$ being left-closed we obtain $u \leq_{\tau_{i,h}} w$. (The base cases for the induction involve shared vertices and thereby only two input graphs.)

size (*chain-bar pair*): If we have $x \leq_{\alpha} w_j$, then there is a shared vertex $x \leq_{\alpha} s \leq_{\alpha} w_j$ and by Theorem 10 we obtain $us \in F$ and $vs \in E$, which yields a chain-bar pair; see Figure 8b. Otherwise we have a chain-bar pair for x and w_j that can be extended by u and v ; see Figure 8c. With the absence of conflicts this ensures that vertices ordered according to the left-closed property are actually distinct.

Assume a new extension would violate the property of antisymmetry. This would mean we already had $v_j <_{\sigma} u_i$, which would imply a cyclic ordering of x, w_j with elements of the (necessary) chain-bar pair for v_j, u_i in a prior step; see Figure 9a. Finally, for more than two input graphs we obtain a corresponding ordering $\tau_{i,j}$ for each pair of input graphs G_i, G_j . Let $\tau = \bigcup_{i,j \in \{1, \dots, k\}} \tau_{i,j}$ be their union. For $u <_{\tau_{i,j}} v <_{\tau_{j,h}} w$ we can prove $u <_{\tau_{i,h}} w$ by using chain-bar pairs and induction; see Figure 9b. Hence, τ is already transitive and the other properties are easy to verify. \blacktriangleleft

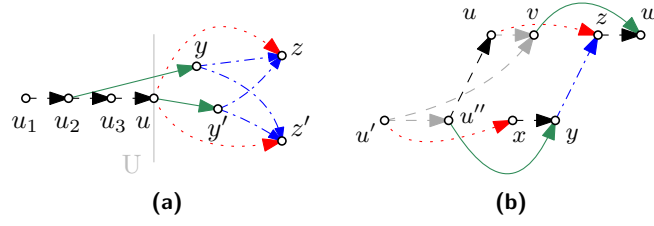
By respecting the orderings obtained by scouting we avoid wrong decisions when greedily adding vertices to a linear ordering in the zipping step; see Figure 7.

Lemma 15. *Let \mathcal{G} be a sunflower graph with a simultaneous enumeration ζ . There is a left-closed linear order τ that extends the partial order α on $V(\mathcal{G})$ induced by ζ if and only if there is a left-closed partial order $\sigma \supseteq \alpha$.*

Proof Sketch. Given σ we process the vertices from the left to the right. We add in each step a leftmost vertex u of the remaining vertices to a set U of the processed vertices that are linearly ordered. We denote the current order by σ' . Vertex u is then ordered before all other vertices in $V \setminus U$. To avoid that the left-closed property is violated when adding such orderings for another vertex, we ensure our extended order $\sigma'' \supseteq \sigma'$ is *right-closed on U* meaning that

$$\forall u, v \in U, w, x \in V: (vw \in E \wedge ux \in F \wedge v \leq u) \Rightarrow w < x. \quad (2)$$

To this end, we consider the current vertex u as vertex u in the definition of right-closed and add all implied orderings in σ'' . This means for each vertex $y \in Y = \{y \in V \mid \exists u' \in U: uy \in E\}$ and each vertex $z \in Z = \{z \in V \mid uz \in F\}$ we set $y \leq_{\sigma''} z$; see Figure 10a. We further extend σ'' to be transitive. Note that there are no two vertices $y \in Y, z \in Z$ with $y \leq_{\sigma} z$, since σ is left-closed and for $u' \in U$ we have $u' \leq_{\sigma} u$. With this observation we can verify that σ'' is antisymmetric and left-closed; see Figure 10b. \blacktriangleleft



■ **Figure 10** (a): orderings added during a zipping step (blue dash-dotted). All vertices in $Y = \{y, y'\}$ are ordered before those in $Z = \{z, z'\}$. (b): The case for σ'' being left-closed where we have $x \leq_{\sigma''} w$ due to transitivity. This means there is some ordering $(y, z) \in Y \times Z$ with $x \leq_{\sigma'} y \leq_{\sigma''} z \leq_{\sigma'} w$. We further have a vertex $u'' \in U$ with $u''y \in E$ and $uz \in F$. Given vertices $u', v \in V$ with $u'x \in F$ and $vw \in E$ we obtain $u' <_{\sigma'} u''$ and $u <_{\sigma'} v$ since σ' is left-closed. This yields $u' <_{\sigma''} v$.

Finally, we construct a graph $H = (V, E')$ for which the obtained linear order τ is a fine enumeration. We do so by setting $E' = \{ux \in V^2 \mid \exists vw \in E: v \leq_{\tau} u <_{\tau} x \leq_{\tau} w\}$ in accordance with Corollary 12 (i).

► **Lemma 16** (*). *Let $\mathcal{G} = (G_1, \dots, G_k)$ be a sunflower graph with a simultaneous enumeration ζ . A linear order τ that extends the partial order on $V(\mathcal{G})$ induced by ζ is a fine enumeration for some graph H that has G_1, \dots, G_k as induced subgraphs if and only if τ is left-closed.*

Combining Lemmas 9, 14, 15 and 16 we obtain Theorem 13.

► **Theorem 13.** *Let \mathcal{G} be a sunflower proper interval graph with simultaneous enumeration ζ . Then \mathcal{G} has a simultaneous unit interval representation that realizes ζ if and only if \mathcal{G} has no conflict for ζ .*

4.2 Recognizing Simultaneous Unit Interval Graphs in Polynomial Time

With Theorems 8 and 13 we can now efficiently recognize simultaneous unit interval graphs.

► **Theorem 17.** *Given a sunflower graph $\mathcal{G} = (G_1, \dots, G_k)$, we can decide in $O(|V| \cdot |E|)$ time, whether \mathcal{G} is a simultaneous unit interval graph, where $(V, E) = G^* = G_1 \cup \dots \cup G_k$. If it is, then we also provide a simultaneous unit interval representation in the same time.*

Proof Sketch. Here we establish polynomial time recognition, and the stated time is proven in the full version. As discussed earlier, we can assume that G^* is connected. With Theorem 6 we obtain a simultaneous enumeration ζ of \mathcal{G} , unless \mathcal{G} is not a simultaneous proper interval graph. By Theorem 13, the sunflower graph \mathcal{G} is a simultaneous unit interval graph if and only if there is a simultaneous enumeration η for which \mathcal{G} has no conflict. In that case η^r also has no conflict. With Theorem 8 we have that η or η^r is obtained from ζ by reversals of reversible parts and independent components. Hence, we only need to consider simultaneous enumerations obtained that way.

Since every single graph G_i is proper, it has no conflict and we only need to consider (u, v) -conflicts with $u, v \in V(S)$, where S is the shared graph. The minimal (u, v) -chains for G_i are exactly the shortest paths in G_i and thus independent from reversals. On the other hand, for the maximal size of (u, v) -bars in G_i only the reversals of the two corresponding components C, D of u, v are relevant, while components in-between always contribute their maximum independent set regardless of whether they are reversed. We can thus compute

for $i, j \in \{1, \dots, k\}$, $u, v \in V(S)$ and each of the four combinations of reversal decisions (reverse or do not reverse) for the corresponding components C, D of u, v , whether they yield a conflict at (u, v) . We can formulate a corresponding 2-SAT formula \mathcal{F} : For every independent component and every reversible part we introduce a literal that represents whether it is reversed or not. For every combination of two reversal decisions that yields a conflict we add a clause that excludes this combination. If \mathcal{F} is not satisfiable, then every simultaneous enumeration yields a conflict. Otherwise, a solution yields a simultaneous enumeration without conflict. We obtain a simultaneous unit interval representation by following the construction in Section 4.1. ◀

5 Conclusion

We studied the problem of simultaneous representations of proper and unit interval graphs. We have shown that, in the sunflower case, both simultaneous proper interval graphs and simultaneous unit intervals can be recognized efficiently. While the former can be recognized by a simple and straightforward recognition algorithm, the latter is based on the three ingredients: 1) a complete characterization of all simultaneous proper interval representations of a sunflower simultaneous graph, 2) a characterization of the simultaneous proper interval representations that can be realized by a simultaneous unit interval representation and 3) an algorithm for testing whether among the simultaneous proper interval representations there is one that satisfies this property.

Future Work. While our algorithm for (sunflower) simultaneous proper interval graphs has optimal linear running time, we leave it as an open problem whether simultaneous unit interval graphs can also be recognized in linear time.

Our main open question is about the complexity of sunflower simultaneous interval graphs. Jampani and Lubiw [18] conjecture that they can be recognized in polynomial time for any number of input graphs. However, even for three graphs the problem is still open.

References

- 1 Patrizio Angelini, Giordano Da Lozzo, and Daniel Neuwirth. On Some \mathcal{NP} -complete SEFE Problems. In Sudebkumar Prasant Pal and Kunihiko Sadakane, editors, *Algorithms and Computation: 8th International Workshop, WALCOM 2014, Chennai, Proceedings*, pages 200–212. Springer, 2014. doi:10.1007/978-3-319-04657-0_20.
- 2 Thomas Bläsius, Stephen G. Kobourov, and Ignaz Rutter. Simultaneous Embedding of Planar Graphs. *CoRR*, abs/1204.5853, 2012. arXiv:1204.5853.
- 3 Thomas Bläsius and Ignaz Rutter. Simultaneous PQ-Ordering with Applications to Constrained Embedding Problems. *ACM Trans. Algorithms*, 12(2):16:1–16:46, 2015. doi:10.1145/2738054.
- 4 Jan Bok and Nikola Jedličková. A note on simultaneous representation problem for interval and circular-arc graphs. *arXiv preprint*, 2018. arXiv:1811.04062.
- 5 Kellogg S. Booth. *PQ Tree Algorithms*. PhD thesis, University of California, Berkeley, 1975.
- 6 Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.
- 7 Peter Brass, Eowyn Cenek, Cristian A Duncan, Alon Efrat, Cesim Erten, Dan P Ismailescu, Stephen G Kobourov, Anna Lubiw, and Joseph SB Mitchell. On simultaneous planar graph embeddings. *Computational Geometry*, 36(2):117–130, 2007. doi:10.1016/j.comgeo.2006.05.006.

- 8 Steven Chaplick, Radoslav Fulek, and Pavel Klavík. Extending Partial Representations of Circle Graphs. In Stephen Wismath and Alexander Wolff, editors, *Graph Drawing: 21st International Symposium, GD 2013, Bordeaux, Revised Selected Papers*, pages 131–142. Springer, 2013. doi:10.1007/978-3-319-03841-4_12.
- 9 Celina M. Herrera de Figueiredo, João Meidanis, and Célia Picinin de Mello. A linear-time algorithm for proper interval graph recognition. *Information Processing Letters*, 56(3):179–184, 1995. doi:10.1016/0020-0190(95)00133-W.
- 10 Xiaotie Deng, Pavol Hell, and Jing Huang. Linear-Time Representation Algorithms for Proper Circular-Arc Graphs and Proper Interval Graphs. *SIAM J. Comput.*, 25(2):390–403, 1996. doi:10.1137/S0097539792269095.
- 11 Alejandro Estrella-Balderrama, Elisabeth Gassner, Michael Jünger, Merijam Percan, Marcus Schaefer, and Michael Schulz. Simultaneous Geometric Graph Embeddings. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *Graph Drawing: 15th International Symposium, GD 2007, Sydney. Revised Papers*, pages 280–290. Springer, 2008. doi:10.1007/978-3-540-77537-9_28.
- 12 Elisabeth Gassner, Michael Jünger, Merijam Percan, Marcus Schaefer, and Michael Schulz. Simultaneous Graph Embeddings with Fixed Edges. In Fedor V. Fomin, editor, *Graph-Theoretic Concepts in Computer Science: 32nd International Workshop, WG 2006, Bergen, Revised Papers*, pages 325–335. Springer, 2006. doi:10.1007/11917496_29.
- 13 Paul W. Goldberg, Martin C. Golumbic, Haim Kaplan, and Ron Shamir. Four strikes against physical mapping of DNA. *Journal of Computational Biology*, 2(1):139–152, 1995. doi:10.1089/cmb.1995.2.139.
- 14 Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57)*. North-Holland Publishing Co., 2004.
- 15 Martin Charles Golumbic, Haim Kaplan, and Ron Shamir. Graph sandwich problems. *Journal of Algorithms*, 19(3):449–473, 1995. doi:10.1006/jagm.1995.1047.
- 16 Pinar Heggernes, Pim van 't Hof, Daniel Meister, and Yngve Villanger. Induced Subgraph Isomorphism on proper interval and bipartite permutation graphs. *Theoretical Computer Science*, 562:252–269, 2015. doi:10.1016/j.tcs.2014.10.002.
- 17 Pavol Hell, Ron Shamir, and Roded Sharan. A Fully Dynamic Algorithm for Recognizing and Representing Proper Interval Graphs. *SIAM J. Comput.*, 31(1):289–305, 2002. doi:10.1137/S0097539700372216.
- 18 Krishnam Raju Jampani and Anna Lubiw. Simultaneous Interval Graphs. In Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park, editors, *Algorithms and Computation: 21st International Symposium, ISAAC 2010, Jeju Island, Proceedings, Part I*, pages 206–217. Springer, 2010. doi:10.1007/978-3-642-17517-6_20.
- 19 Krishnam Raju Jampani and Anna Lubiw. The Simultaneous Representation Problem for Chordal, Comparability and Permutation Graphs. *Journal of Graph Algorithms and Applications*, 16(2):283–315, 2012. doi:10.7155/jgaa.00259.
- 20 Pavel Klavík, Jan Kratochvíl, Yota Otachi, Ignaz Rutter, Toshiki Saitoh, Maria Saumell, and Tomáš Vyskočil. Extending Partial Representations of Proper and Unit Interval Graphs. *Algorithmica*, 77(4):1071–1104, April 2017. doi:10.1007/s00453-016-0133-z.
- 21 Peter J Looges and Stephan Olariu. Optimal greedy algorithms for indifference graphs. *Computers & Mathematics with Applications*, 25(7):15–25, 1993. doi:10.1016/0898-1221(93)90308-I.
- 22 Ross M McConnell and Yahav Nussbaum. Linear-time recognition of probe interval graphs. In Amos Fiat and Peter Sanders, editors, *Proceedings of the 17th Annual European Symposium on Algorithms (ESA'09)*, volume 5757 of *Lecture Notes in Computer Science*, pages 349–360. Springer, 2009. doi:10.1007/978-3-642-04128-0_32.
- 23 Yahav Nussbaum. Recognition of probe proper interval graphs. *Discrete Applied Mathematics*, 167:228–238, 2014. doi:10.1016/j.dam.2013.11.013.

- 24 Fred S Roberts. *Representations of indifference relations*. PhD thesis, Department of Mathematics, Stanford University, 1968.
- 25 Fred S. Roberts. Indifference Graphs. In F. Harary, editor, *Proof Techniques in Graph Theory*, pages 139–146. Academic Press, New York, 1969.
- 26 Marcus Schaefer. Toward a Theory of Planarity: Hanani-Tutte and Planarity Variants. In Walter Didimo and Maurizio Patrignani, editors, *Graph Drawing: 20th International Symposium, GD 2012, Redmond, Revised Selected Papers*, pages 162–173. Springer, 2013. doi:10.1007/978-3-642-36763-2_15.
- 27 Dale J. Skrien. A relationship between triangulated graphs, comparability graphs, proper interval graphs, proper circular-arc graphs, and nested interval graphs. *Journal of Graph Theory*, 6(3):309–316, 1982. doi:10.1002/jgt.3190060307.
- 28 Jeremy P. Spinrad. *Efficient Graph Representations*. Fields Institute Monographs. AMS, 2003.

Correlation Clustering with Same-Cluster Queries Bounded by Optimal Cost

Barna Saha

University of California, Berkeley, USA

<http://www.barnasaha.net>

barnas@berkeley.edu

Sanjay Subramanian

Allen Institute for Artificial Intelligence, Seattle, WA, USA¹

<http://www.sanjayss34.github.io>

sanjays@allenai.org

Abstract

Several clustering frameworks with interactive (semi-supervised) queries have been studied in the past. Recently, clustering with same-cluster queries has become popular. An algorithm in this setting has access to an oracle with full knowledge of an optimal clustering, and the algorithm can ask the oracle queries of the form, “Does the optimal clustering put vertices u and v in the same cluster?” Due to its simplicity, this querying model can easily be implemented in real crowd-sourcing platforms and has attracted a lot of recent work.

In this paper, we study the popular correlation clustering problem (Bansal et al., 2002) under the same-cluster querying framework. Given a complete graph $G = (V, E)$ with positive and negative edge labels, correlation clustering objective aims to compute a graph clustering that minimizes the total number of disagreements, that is the negative intra-cluster edges and positive inter-cluster edges. In a recent work, Ailon et al. (2018b) provided an approximation algorithm for correlation clustering that approximates the correlation clustering objective within $(1 + \epsilon)$ with $O(\frac{k^{14} \log n \log k}{\epsilon^6})$ queries when the number of clusters, k , is fixed. For many applications, k is not fixed and can grow with $|V|$. Moreover, the dependency of k^{14} on query complexity renders the algorithm impractical even for datasets with small values of k .

In this paper, we take a different approach. Let C_{OPT} be the number of disagreements made by the optimal clustering. We present algorithms for correlation clustering whose error and query bounds are parameterized by C_{OPT} rather than by the number of clusters. Indeed, a good clustering must have small C_{OPT} . Specifically, we present an efficient algorithm that recovers an exact optimal clustering using at most $2C_{OPT}$ queries and an efficient algorithm that outputs a 2-approximation using at most C_{OPT} queries. In addition, we show under a plausible complexity assumption, there does not exist any polynomial time algorithm that has an approximation ratio better than $1 + \alpha$ for an absolute constant $\alpha > 0$ with $o(C_{OPT})$ queries. Therefore, our first algorithm achieves the optimal query bound within a factor of 2.

We extensively evaluate our methods on several synthetic and real-world datasets using real crowd-sourced oracles. Moreover, we compare our approach against known correlation clustering algorithms that do not perform querying. In all cases, our algorithms exhibit superior performance.

2012 ACM Subject Classification Theory of computation → Unsupervised learning and clustering; Theory of computation → Approximation algorithms analysis

Keywords and phrases Clustering, Approximation Algorithm, Crowdsourcing, Randomized Algorithm

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.81

Related Version A full version of the paper is available at <https://arxiv.org/abs/1908.04976>.

Supplement Material Code and Data:

<https://www.github.com/sanjayss34/corr-clust-query-esa2019>

¹ Most of this work was completed when the second author was at the University of Pennsylvania and the University of Massachusetts at Amherst.



Funding *Barna Saha*: B. Saha is partially supported by an NSF CAREER Award CCF 1652303, a Google Faculty Award and an Alfred P. Sloan fellowship.

Sanjay Subramanian: This work was supported in part by the National Science Foundation (NSF) Research Experiences for Undergraduates (REU) program.

Acknowledgements The second author would like to thank Dan Roth for letting him use his machines for running experiments, Sainyam Galhotra for help with datasets, and Rajiv Gandhi for useful discussions.

1 Introduction

In correlation clustering, the algorithm is given potentially inconsistent information about similarities and dissimilarities between pairs of vertices in a graph, and the task is to cluster the vertices so as to minimize disagreements with the given information [7, 10]. The correlation clustering problem was first proposed by Bansal, Blum and Chawla [7] and since then it has found numerous applications in document clustering, image segmentation, grouping gene expressions etc. [7, 10].

In correlation clustering, we are given a complete graph $G = (V, E)$, $|V| = n$, where each edge is labelled either $+$ or $-$. An optimal clustering partitions the vertices such that the number of intra-cluster negative edges and inter-cluster positive edges is minimized. The problem is known to be NP-Hard. The seminal work of Bansal et al. [7] gave a constant factor approximation for correlation clustering. Following a long series of works [7, 9, 4, 16, 12], the best known approximation bounds till date are a 3-approximation combinatorial algorithm [1] and a 2.06-approximation based on linear programming rounding [10]. The proposed linear programming relaxation for correlation clustering [9, 1, 10] is known to have an integrality gap of 2, but there does not exist yet a matching algorithm that has an approximation ratio 2 or lower.

Correlation clustering problem can be extended to weighted graphs for an $O(\log n)$ -approximation bound and is known to be optimal [12]. Moreover, when one is interested in maximizing agreements, a polynomial time approximation scheme was provided by Bansal et al. [7].

Over the last two decades, crowdsourcing has become a widely used way to generate labeled data for supervised learning. The same platforms that are used for this purpose can also be used for unsupervised problems, thus converting the problems to a semi-supervised active learning setting. This can often lead to significant improvements in accuracy. However, using crowdsourcing introduces another dimension to the optimization problems, namely minimizing the amount of crowdsourcing that is used. The setting of active querying has been studied previously in the context of various clustering problems. Balcan and Blum [6] study a clustering problem in which the only information given to the algorithm is provided through an oracle that tells the algorithm either to “merge” two clusters or to “split” a cluster. More recently, Ashtiani, Kushgrha and Ben-David [5] considered a framework of same-cluster queries for clustering; in this framework, the algorithm can access an oracle that has full knowledge of an optimal clustering and can issue queries to the oracle of the form “Does the optimal clustering put vertices u and v in the same cluster?” Because of its simplicity, such queries are highly suited for crowdsourcing and has been studied extensively both in theory community [3, 19, 2, 15] and in applied domains [23, 14, 17, 22]. Correlation clustering has also been considered in this context. Ailon, Bhattacharya and Jaiswal [2] study correlation clustering in this framework under the assumption that the number k of clusters is fixed. They gave an $(1 + \epsilon)$ approximation algorithm for correlation clustering

that runs in polynomial time and issues $O(k^{14} \log n \log k / \epsilon^6)$ queries. However, for most relevant applications, the number of clusters k is not fixed. Even for fixed k , the dependence of k^{14} is huge (consider $k = 2$ and $2^{14} = 16384$ with additional constants terms hidden under $O()$ notation).

In this paper, we give near-optimal algorithms for correlation clustering with same-cluster queries that are highly suitable for practical implementation and whose performance is parameterized by the optimum number of disagreements. Along with providing theoretical guarantees, we perform extensive experiments on multiple synthetic and real datasets. Let C_{OPT} be the number of disagreements made by the optimal clustering. Our contributions are as follows.

1. A deterministic algorithm that outputs an *optimal* clustering using at most $2C_{OPT}$ queries (Section 3).
2. An expected 2-approximation algorithm that uses at most C_{OPT} queries in expectation (Section 4).
3. A new lower bound that shows it is not possible to get an $(1 + \alpha)$ approximation for some constant $\alpha > 0$ with any polynomial time algorithm that issues $o(C_{OPT})$ queries assuming GAP-ETH (see definition in Section 5).
4. An extensive experimental comparison that not only compares the effectiveness of our algorithms, but also compares the state-of-the art correlation clustering algorithms that do not require any querying (Section 6).

Assumption of an optimum oracle [5, 2] is quite strong in practice. However, our experiments reveal that such an assumption is not required. In correlation clustering, often the \pm edges are generated by fitting an automated classifier, where each vertex corresponds to some object and is associated with a feature vector. In our experiments with real-world data, instead of an optimum oracle, we use crowdsourcing. By making only a few pair-wise queries to a crowd oracle, we show it is possible to obtain an optimum or close to optimum clustering. After our work, it came to our notice that it may be possible to use Bocker et al.'s [8] results on fixed-parameter trackability of cluster editing to adapt to our setting, and get better constants on the query complexity. This is an ongoing work. Our algorithms and techniques are vastly different from [8] and are also considerably simpler.

2 Related Work

Asthiani et al. [5] considered the k -means objective with same-cluster queries and showed that it is possible to recover the optimal clustering under k -means objective with high probability by issuing $O(k^2 \log k + k \log n)$ queries if a certain margin condition holds for each cluster. Gamlath, Huang and Svensson extended the above result when approximation is allowed [15]. Ailon et al. [2] studied correlation clustering with same-cluster queries and showed that there exists an $(1 + \epsilon)$ approximation for correlation clustering where the number of queries is a (large) polynomial in k . Our algorithms are different from those in [2] in that our guarantees are parameterized by C_{OPT} rather than by k . Kushagra et al. [18] study a restricted version of correlation clustering where the valid clusterings are provided by a set of hierarchical trees and provide an algorithm using same-cluster queries for a related setting, giving guarantees in terms of the size of the input instance (or the VC dimension of the input instance) rather than C_{OPT} . [19] studied, among other clustering problems, a random

instance of correlation clustering under same-cluster queries. Our algorithms are based on the basic 3-approximation algorithm of Ailon et al. [1] that selects a pivot vertex randomly and forms a cluster from that vertex and all of its $+$ -neighbors. They further honed this approach by choosing to keep each vertex in the pivot's cluster with a probability that is a function of the linear programming solution. Chawla et al. [10] used a more sophisticated function of the linear programming solution to design the current state-of-the-art algorithm, which gives a 2.06 approximation for correlation clustering.

3 Finding an Optimal Clustering

We are given a query access to an oracle that given any two vertices u and v returns whether or not u and v are together in a cluster in an optimal solution. Let OPT denote the optimal solution which is used by the oracle. Given a positive ($+$) edge (u, v) , if OPT puts u and v in different clusters, then we say OPT makes a mistake on that edge. Similarly for a negative ($-$) edge (u, v) , if OPT puts them together in a cluster then again OPT makes a mistake on it. Similarly, our algorithm can decide to make mistakes on certain edges and our goal is to minimize the overall number of mistakes. It is easy to see that an optimal solution for a given input graph makes mistakes only on edges that are part of a $(+, +, -)$ triangle. Moreover, any optimal solution must make at least one mistake in such a triangle.

The pseudocode for our algorithm, QUERYPIVOT, is given in Algorithm 1. The algorithm is as follows (in the following description, we give in brackets the corresponding line number for each step). We pick a pivot u arbitrarily from the set of vertices that are not clustered yet [line 5]. For each $(+, +, -)$ triangle (u, v, w) [line 10], if we have not yet determined via queries that OPT makes a mistake on $\{u, v\}$ or that OPT makes a mistake on $\{u, w\}$ [lines 11-14], then (1) we query $\{u, v\}$ [line 17] and if OPT makes a mistake on this edge, we too decide to make a mistake on this edge and proceed to the next $(+, +, -)$ triangle involving u and (2) if OPT does not make a mistake on $\{u, v\}$, then we query $\{u, w\}$ [line 23] and make a mistake on it if OPT makes a mistake on it. Note that if we have already queried one of $\{u, v\}$ or $\{u, w\}$ and found a mistake, we do not query the other edge [line 11]. Once we have gone through all $(+, +, -)$ triangles involving u then for every $v \neq u$, if we have not already decided to make a mistake on $\{u, v\}$, then if $\{u, v\}$ is a $+$ edge we keep v in u 's cluster and if $\{u, v\}$ is a $-$ edge we do not put v in u 's cluster. On the other hand, if we have decided to make a mistake on $\{u, v\}$, then if $\{u, v\}$ is a $-$ edge we keep v in u 's cluster and if $\{u, v\}$ is a $+$ edge we do not put v in u 's cluster. Finally, we remove all vertices in u 's cluster from the set of remaining vertices and recursively call the function on the set of remaining vertices.

In the pseudocode, $Queried[v] = 1$ means the algorithm has already issued a query $(pivot, v)$ to the oracle, $Mistake[v] = 1$ means it has decided to make a mistake on the edge $(pivot, v)$ based on the oracle answer, and $Oracle(pivot, v)$ returns 1 iff OPT makes a mistake on the edge $\{pivot, v\}$. We prove the following theorem that shows that QUERYPIVOT is able to recover the optimal clustering known to the oracle with a number of queries bounded in terms of C_{OPT} .

► **Theorem 3.1.** *Let C_{OPT} be the number of mistakes made by an optimal clustering. The QUERYPIVOT algorithm makes C_{OPT} mistakes and makes at most $2C_{OPT}$ queries to the oracle.*

Algorithm 1 QUERYPIVOT.

```

1: Input: vertex set  $V$ , adjacency matrix  $A$ , oracle  $Oracle$ 
2: if  $V == \emptyset$  then
3:   return  $\emptyset$ 
4: end if
5:  $pivot \leftarrow$  Arbitrary vertex in  $V$ 
6:  $T \leftarrow$  all  $(+, +, -)$  triangles that include  $pivot$ 
7:  $C \leftarrow V$ 
8:  $Queried \leftarrow$  length- $n$  array of zeros
9:  $Mistakes \leftarrow$  length- $n$  array of zeros
10: for  $(pivot, v, w) \in T$  do
11:   if  $Mistake[v] == 1$  or  $Mistake[w] == 1$  then
12:     continue
13:   else if  $Queried[v] == 1$  and  $Queried[w] == 1$  then
14:     continue
15:   else if  $Queried[v] == 0$  then
16:      $Queried[v] \leftarrow 1$ 
17:     if  $Oracle(pivot, v) == 1$  then
18:        $Mistake[v] \leftarrow 1$ 
19:     end if
20:   end if
21:   if  $Queried[w] == 0$  and  $Mistake[v] == 0$  then
22:      $Queried[w] \leftarrow 1$ 
23:     if  $Oracle(pivot, w) == 1$  then
24:        $Mistake[w] \leftarrow 1$ 
25:     end if
26:   end if
27: end for
28: for  $v \in V \setminus \{pivot\}$  do
29:   if  $(v \in N^-(pivot)$  and  $Mistake[v] == 0)$  or  $(v \in N^+(pivot)$  and  $Mistake[v] == 1)$  then
30:      $C = C \setminus \{v\}$ 
31:   end if
32: end for
33: return  $\{C\} \cup \text{QUERYPIVOT}(V \setminus C, A, Oracle)$ 

```

For a given cluster C and a vertex $w \in C$, we denote by $N_C^+(w)$ the set of vertices in C that have $+$ edges with w . Similarly, we denote by $N_C^-(w)$ the set of vertices in C that have $-$ edges.

The algorithm time complexity is dominated by the time taken to check $(+, +, -)$ triangles involved with the pivots. Let E^+ denote the set of positive edges in G . Then all the $(+, +, -)$ triangles that include a pivot can be checked in time $O(|E^+| * n)$.

► **Lemma 3.2.** *The QUERYPIVOT algorithm outputs a valid partition of the vertices.*

Proof. Note that the pivot is never removed from C . Hence, between each pair of consecutive recursive calls, at least one vertex is removed from V . The algorithm must then terminate after at most n recursive calls. Moreover, in each recursive call, the set of vertices passed to the next recursive call is disjoint from the cluster created in that recursive call. Thus, inductively, the sets returned by the algorithm must be disjoint. ◀

► **Lemma 3.3.** *Consider a clustering \mathcal{C} in which some cluster C contains vertices u, v s.t. $\{u, v\}$ is a $-$ edge and s.t. u and v do not form a $(+, +, -)$ triangle with any other vertex in C . \mathcal{C} is suboptimal.*

Proof. If we were to remove v from C and put it in a singleton cluster, we would make $|N_C^-(v)| - |N_C^+(v)|$ fewer mistakes than C . If $|N_C^-(v)| - |N_C^+(v)| > 0$, then C is suboptimal. Therefore, assume $|N_C^-(v)| \leq |N_C^+(v)|$. Now, note that $\forall w \in N_C^+(v)$, $w \in N_C^-(u)$ because otherwise, u , v , and w form a $(+, +, -)$ triangle. Thus $|N_C^+(v)| \leq |N_C^-(u)| - 1$ because $v \in N_C^-(u)$. Moreover, $|N_C^+(u)| \leq |N_C^-(v)| - 1$ because $u \in N_C^-(v)$.

Hence, if we were to remove u from C and put it in a singleton cluster, we would make $|N_C^-(u)| - |N_C^+(u)| \geq |N_C^+(v)| - |N_C^-(v)| + 2$ fewer mistakes than C . Since $|N_C^-(v)| - |N_C^+(v)| \leq 0$, $|N_C^+(v)| - |N_C^-(v)| + 2 > 0$, so C is suboptimal. \blacktriangleleft

► **Lemma 3.4.** *Consider a clustering \mathcal{C} in which a cluster C_1 contains a vertex u , a different cluster C_2 contains a vertex v , $\{u, v\}$ is a $+$ edge, and in every $(+, +, -)$ triangle that includes $\{u, v\}$, the clustering makes at least 2 edge mistakes. \mathcal{C} is suboptimal.*

Proof. If we were to remove u from C_1 and put it in C_2 , we would make $|N_{C_2}^+(u)| + |N_{C_1}^-(u)| - |N_{C_2}^-(u)| - |N_{C_1}^+(u)| = 2|N_{C_2}^+(u)| + |C_1| - |C_2| - 2|N_{C_1}^+(u)|$ fewer mistakes than C . If $2|N_{C_2}^+(u)| + |C_1| - |C_2| - 2|N_{C_1}^+(u)| > 0$, then C is suboptimal. Otherwise, note that $\forall w \in N_{C_1}^+(u)$, $w \in N_{C_1}^-(v)$ because if not, u, v, w would form a $(+, +, -)$ triangle in which the algorithm makes fewer than 2 edge mistakes. By a similar argument, $\forall w \in N_{C_2}^+(v)$, $w \in N_{C_2}^-(u)$. Thus, since in addition, $\{u, v\}$ is a $+$ edge, we have that $|N_{C_1}^+(v)| \geq |N_{C_1}^+(u)| + 1$ and $|N_{C_2}^+(u)| \geq |N_{C_2}^+(v)| + 1$. Now if we were to remove v from C_2 and put it in C_1 , the number of mistakes will reduce by $|N_{C_1}^+(v)| + |N_{C_2}^-(v)| - |N_{C_1}^-(v)| - |N_{C_2}^+(v)| = 2|N_{C_1}^+(v)| + |C_2| - |C_1| - 2|N_{C_2}^+(v)|$. Since $|N_{C_1}^+(v)| \geq |N_{C_1}^+(u)| + 1$ and $|N_{C_2}^+(v)| \leq |N_{C_2}^+(u)| - 1$, we have that $2|N_{C_1}^+(v)| + |C_2| - |C_1| - 2|N_{C_2}^+(v)| \geq 2(|N_{C_1}^+(u)| + 1) + |C_2| - |C_1| - 2(|N_{C_2}^+(u)| - 1)$. Since $2|N_{C_2}^+(u)| + |C_1| - |C_2| - 2|N_{C_1}^+(u)| \leq 0$, $2(|N_{C_1}^+(u)| + 1) + |C_2| - |C_1| - 2(|N_{C_2}^+(u)| - 1) > 0$, so C is suboptimal. \blacktriangleleft

► **Lemma 3.5.** *When given an oracle corresponding to an optimal clustering OPT , the clustering returned by the QUERYPIVOT algorithm is identical to OPT . It follows that the algorithm's clustering makes at most as many mistakes as OPT .*

Proof. We will prove inductively that in each recursive call, the cluster C returned by the algorithm is a cluster in OPT . Note that at the beginning of the first recursive call, the claim that all clusters formed so far are clusters in OPT is vacuously true because there are no clusters yet formed. Now consider an arbitrary but particular recursive call, and let u be the pivot in this recursive call. Suppose for contradiction that C is not a cluster in OPT .

Case 1: There is a vertex v such that $v \notin C$, but in OPT , v is in the same cluster as u . Let H be the cluster in OPT that contains u and v . First, observe that H must be a subset of the remaining vertices in this recursive call; otherwise, one of the clusters formed in a previous call contains some vertex in H but does not include u , contradicting the induction hypothesis because this previously formed cluster is not a cluster in OPT . Next, note that for any mistake that the algorithm makes on an edge incident on a pivot, the algorithm queries the OPT oracle and makes the mistake iff OPT makes the mistake. Then if $\{u, v\}$ is a $+$ edge, then the algorithm must have queried the oracle for $\{u, v\}$ and found that OPT makes a mistake on it because the algorithm decided to make a mistake on that edge. This implies that OPT puts u and v in different clusters, which is a contradiction. Now suppose instead that $\{u, v\}$ is a $-$ edge. Again if the algorithm queried the oracle for $\{u, v\}$, then OPT must have put u and v in different clusters, so it must be the case that the algorithm did not query the oracle for $\{u, v\}$. It follows that for any $(+, +, -)$ triangle (u, v, w) that includes $\{u, v\}$, our algorithm has queried $\{u, w\}$ and found OPT makes a mistake on the $+$ edge $\{u, w\}$. Then for any such triangle, $w \notin H$. It follows that u and v do not form a

$(+, +, -)$ triangle with any vertex in H . Since u and v are in the same cluster H in OPT , $\{u, v\}$ is a $-$ edge, and u and v do not form a $(+, +, -)$ triangle with any other vertex in H , the conditions for Lemma 3.3 are satisfied. Therefore, OPT is a suboptimal clustering, which is a contradiction.

Case 2: There is a vertex v such that $v \in C$, but in OPT , v is not in u 's cluster. As in the first case, if $\{u, v\}$ were a $-$ edge, the algorithm must make a mistake on $\{u, v\}$ and so must have queried OPT and found that OPT made a mistake on $\{u, v\}$, a contradiction. Now suppose instead that $\{u, v\}$ is a $+$ edge. If there is some vertex w that was clustered prior to this recursive call s.t. $\{u, v, w\}$ is a $(+, +, -)$ triangle in which OPT makes exactly one mistake (on $\{u, v\}$), then note that either u or v should be in the same cluster as w because one of $\{u, w\}$ and $\{v, w\}$ must be a $+$ edge; in this case, we have reached a contradiction with the inductive hypothesis because the previously formed cluster that included w did not include u or v . Then in order to show that the conditions for Lemma 3.4 are satisfied, we must show that for every vertex w in the set of remaining vertices when u is the pivot, if (u, v, w) is a $(+, +, -)$ triangle, then OPT must make at least two mistakes in the triangle. Since OPT makes a mistake on $\{u, v\}$ but the algorithm does not do so, it must be the case that the algorithm did not query $\{u, v\}$. Since the algorithm did not query $\{u, v\}$, for every $(+, +, -)$ triangle (u, v, w) that includes $\{u, v\}$ and such that w is in the set of remaining vertices when u is the pivot, OPT must make a mistake on $\{u, w\}$. Then since OPT makes a mistake on $\{u, v\}$ and on $\{u, w\}$ in any $(+, +, -)$ triangle (u, v, w) , we have by Lemma 3.4 that OPT is suboptimal clustering, which is a contradiction. \blacktriangleleft

► **Lemma 3.6.** *Let C_{OPT} be the number of mistakes made by an optimal clustering OPT . Then the QUERYPIVOT algorithm makes at most $2C_{OPT}$ queries to the oracle.*

Proof. The algorithm queries the oracle only when considering $(+, +, -)$ triangles. Note that whenever considering a particular $(+, +, -)$ triangle, if the algorithm makes a query, it makes at most two queries when considering that triangle and makes at least one mistake that had not been made when considering previous triangles. Therefore, the algorithm makes at most twice as many queries as mistakes. Since the algorithm makes exactly C_{OPT} mistakes, the algorithm makes at most $2C_{OPT}$ queries. \blacktriangleleft

Theorem 3.1 follows directly from Lemmas 3.5 and 3.6.

4 A 2-Approximation Algorithm for Correlation Clustering

A natural question that arises from QUERYPIVOT is how to use fewer queries and obtain an approximation guarantee that is better than the state-of-the-art outside the setting with same-cluster queries, which is a 2.06-approximation. In this section, we show that a randomized version of QUERYPIVOT gives a 2-approximation in expectation using at most C_{OPT} queries in expectation.

The algorithm RANDOMQUERYPIVOT(p) is as follows. We pick a pivot u uniformly at random from the vertices yet to be clustered. For each $(+, +, -)$ triangle (u, v, w) , we have two cases. (1) If $\{u, v\}$ and $\{u, w\}$ are both $+$ edges, then with probability p (chosen appropriately), we query both $\{u, v\}$ and $\{u, w\}$ and for each of these two edges we make a mistake on the edge iff OPT makes a mistake on the edge. With probability $1 - p$ we make no queries for this triangle and proceed to the next triangle. (2) If one of $\{u, v\}$ and $\{u, w\}$ is a $+$ edge and the other is a $-$ edge, then with probability p , we do the following. First, we query the $+$ edge and if OPT makes a mistake on it, then we make a mistake on it and proceed to the next triangle. If OPT does not make a mistake on the $+$ edge, then we query

Algorithm 2 RANDOMQUERYPIVOT.

```

1: Input: vertex set  $V$ , adjacency matrix  $A$ , oracle  $Oracle$ , parameter  $p$ 
2: if  $V == \emptyset$  then
3:   return  $\emptyset$ 
4: end if
5:  $pivot \leftarrow$  Random vertex in  $V$ 
6:  $T \leftarrow$  all  $(+, +, -)$  triangles that include  $pivot$ 
7:  $C \leftarrow V$ 
8:  $Queried \leftarrow$  length- $n$  array of zeros
9:  $Mistakes \leftarrow$  length- $n$  array of zeros
10: for  $(pivot, v, w) \in T$  do
11:   // Without loss of generality, suppose that  $\{pivot, v\}$  is a  $+$  edge
12:   Sample  $r$  from  $Uniform(0, 1)$ 
13:   if  $r > p$  then
14:     continue
15:   end if
16:   if  $Oracle(pivot, v) == 1$  then
17:      $Mistake[v] \leftarrow 1$ 
18:   end if
19:   if  $Mistake[v] == 0$  or  $\{pivot, w\}$  is a  $+$  edge then
20:     if  $Oracle(pivot, w) == 1$  then
21:        $Mistake[v] \leftarrow 1$ 
22:     end if
23:   end if
24: end for
25: for  $v \in V \setminus \{pivot\}$  do
26:   if  $(v \in N^-(pivot) \text{ and } Mistake[v] == 0)$  or  $(v \in N^+(pivot) \text{ and } Mistake[v] == 1)$  then
27:      $C = C \setminus \{v\}$ 
28:   end if
29: end for
30: return  $\{C\} \cup \text{RANDOMQUERYPIVOT}(V \setminus C, A, Oracle)$ 

```

the $-$ edge and make a mistake on the $-$ edge iff OPT does so. Again, with probability $1 - p$ we make no queries for this triangle and proceed to the next triangle. Once we have gone through all triangles, if we have not already decided to make a mistake on $\{u, v\}$, then if $\{u, v\}$ is a $+$ edge we keep v in u 's cluster and if $\{u, v\}$ is a $-$ edge we do not put v in u 's cluster. On the other hand, if we have decided to make a mistake on $\{u, v\}$, then if $\{u, v\}$ is a $-$ edge we keep v in u 's cluster and if $\{u, v\}$ is a $+$ edge we do not put v in u 's cluster. Finally, we remove all vertices in u 's cluster from the set of remaining vertices and recursively call the function on the set of remaining vertices. Note that given a pivot u and a $(+, +, -)$ triangle containing u , if the algorithm chooses not to query either of the edges incident on u , then the algorithm must make a mistake on the edge opposite to u in that triangle.

► **Theorem 4.1.** $\text{RANDOMQUERYPIVOT}(p)$ gives a $\max\left(2, \frac{3}{1+2p}\right)$ -approximation in expectation and uses at most $\max(4p, 1) * C_{OPT}$ queries in expectation.

► **Corollary 4.2.** When $p = 0.25$, RANDOMQUERYPIVOT gives a 2-approximation in expectation and uses at most C_{OPT} queries in expectation.

► **Lemma 4.3.** In an arbitrary but particular recursive call, the probability that RANDOMQUERYPIVOT queries edge $\{u, v\}$ on which OPT makes a mistake given that u is the pivot is equal to the probability that RANDOMQUERYPIVOT queries edge $\{u, v\}$ given that v is the pivot.

Proof. For a + edge $\{u, v\}$ on which OPT makes a mistake, the probability that the edge is queried given that one of the vertices is the pivot is a function only of the number of $(+, +, -)$ triangles that include the edge. In particular, if T is the number of $(+, +, -)$ triangles including the edge, the probability that the edge is queried is $1 - (1 - p)^T$. This number of triangles does not depend on the pivot vertex, so the claim holds if $\{u, v\}$ is a + edge. If $\{u, v\}$ is a - edge, then we claim that the probability that $\{u, v\}$ is queried given that either u or v is a function only of the number of $(+, +, -)$ triangles that include $\{u, v\}$ in which OPT makes a mistake only on this - edge. This claim is true because (1) in any $(+, +, -)$ triangle in which OPT makes a mistake on the - and a + edge, OPT must make a mistake on all of the three edges in the triangle and (2) when considering a $(+, +, -)$ triangle such that the pivot is an endpoint of the - edge, the algorithm queries the - edge iff OPT does not make a mistake on the + edge of which the pivot is an endpoint. It follows that for any $(+, +, -)$ triangle in which the algorithm queries the - edge, OPT must make a mistake only on the - edge. Since the number of $(+, +, -)$ triangles that include $\{u, v\}$ in which OPT makes a mistake only on $\{u, v\}$ does not depend on whether u or v is the pivot, the claim holds when $\{u, v\}$ is a - edge. ◀

Let $s_{uv} = 1$ if $\{u, v\}$ is a - edge and 0 otherwise. Let c_{uv}^* equal 1 if OPT makes a mistake on $\{u, v\}$ and 0 otherwise.

Let OPT^t be the number of edges $\{u, v\}$ s.t. $c_{uv}^* = 1$ and the algorithm makes a decision on $\{u, v\}$ in iteration t . Let ALG^t be the number of edges $\{u, v\}$ s.t. the algorithm makes a mistake on $\{u, v\}$ and the algorithm makes a decision on $\{u, v\}$ in iteration t .

Let V_t be the set of vertices remaining at the beginning of iteration t . Let D_{uv}^t be the event that the algorithm makes a decision on $\{u, v\}$ in iteration t .

► **Lemma 4.4.** *Let T be the number of iterations that the algorithm takes to cluster all vertices. If $E[ALG^t|V_t] \leq \alpha E[OPT^t|V_t]$, for each iteration t , then $E\left[\sum_{t=1}^T ALG^t\right] \leq \alpha E\left[\sum_{t=1}^T OPT^t\right]$.*

Proof. Define $X_0 = 0$ and for each $s > 0$, define $X_s = \sum_{t=1}^s \alpha OPT^t - ALG^t$. If the condition in the lemma holds, then X_s is a submartingale because $E[X_{s+1}|X_s] \geq X_s$. Also, T is a stopping time that is almost surely bounded (since $T \leq n$ with probability 1). By Doob's optional stopping theorem [24, p. 100], if T is a stopping time that is almost surely bounded and X is a discrete-time submartingale, then $E[X_T] \geq E[X_0]$. Then we have that $E[X_T] = E\left[\sum_{t=1}^T \alpha OPT^t - ALG^t\right] \geq E[X_0] = 0$. ◀

► **Lemma 4.5.** *The expected number of mistakes made by the algorithm's clustering is at most $\max\left(2, \frac{3}{1+2p}\right) C_{OPT}$.*

Proof Sketch. Here we give a sketch of the proof. By Lemma 4.4, if we show that $E[ALG^t - \alpha OPT^t] \leq 0$ for any t , where $\alpha \leq \max\left(2, \frac{3}{1+2p}\right)$, then the claim will follow. Let A_w^t be the event that $w \in V_t$ is the pivot in iteration t .

$$E[OPT^t|V_t] = \sum_{\{u,v\} \subseteq E \cap (V_t \times V_t)} \frac{c_{uv}^*}{|V_t|} \sum_{w \in V_t} \Pr[D_{uv}^t | A_w^t]$$

Now we will write $E[ALG^t|V_t]$ by charging the algorithm's mistakes to each of OPT 's mistakes.

81:10 Same-Cluster Queries Bounded by Optimal Cost

Let M_{uv}^t be the charge incurred to $\{u, v\}$ in iteration t . We will assign charges such that $M_{uv}^t = 0$ if $c_{uv}^* = 0$. Then

$$E[ALG^t|V_t] = \sum_{\{u,v\} \subseteq E \cap (V_t \times V_t)} c_{uv}^* E[M_{uv}^t] = \sum_{\{u,v\} \subseteq E \cap (V_t \times V_t)} c_{uv}^* \frac{1}{|V_t|} \sum_{w \in V_t} E[M_{uv}^t | A_w^t]$$

Our goal is to compute an upper bound on $E[M_{uv}^t | A_w^t]$. To do so, we define several events.

For each edge $\{u, v\}$ s.t. $c_{uv}^* = 1$, define the following subsets of V_t : $\{u, v\}$, $\forall i \in \{1, 2, 3\}$, T_i^{uv} is the set of vertices w s.t. $\{u, v, w\}$ is a $(+, +, -)$ triangle in which OPT makes exactly i mistakes, S^{uv} is the set of vertices w s.t. $\{u, v, w\}$ is a $(+, -, -)$ or $(+, +, +)$ triangle in which OPT makes exactly 2 mistakes, $R^{uv} \equiv V_t \setminus T_1^{uv} \setminus T_2^{uv} \setminus S^{uv} \setminus \{u, v\}$. Furthermore, let T_{2u}^{uv} be the subset of T_2^{uv} s.t. $w \in T_{2u}^{uv}$ if the 2 mistakes in $\{u, v, w\}$ are both incident on u . Similarly, let S_u^{uv} be the subset of S^{uv} s.t. $w \in S_u^{uv}$ if the 2 mistakes in $\{u, v, w\}$ are both incident on u . $\forall w \in T_1^{uv}$, the probability that the algorithm makes a mistake on $\{u, v\}$ given that w is the pivot is $\Pr[D_{uv}^t | A_w^t] = 1$. Note that T_1^{uv} , T_2^{uv} , $\{u, v\}$, S^{uv} , and R^{uv} partition V_t .

We compute $E[M_{uv}^t | A_w^t]$ (or an upper bound thereof) when w is in each of the sets $\{u, v\}$, T_1^{uv} , T_{2u}^{uv} , T_{2v}^{uv} , S_u^{uv} , S_v^{uv} , and R^{uv} , which partition V_t . Similarly, we analyze $\Pr[D_{uv}^t | A_w^t]$, breaking up the calculation based on whether the pivot w is in T_1^{uv} , T_{2u}^{uv} , T_{2v}^{uv} , S_u^{uv} , S_v^{uv} , $\{u, v\}$, or R^{uv} .

In order to prove the claim, we show that

$$\sum_{w \in V_t} E[M_{uv}^t | A_w^t] \leq \max\left(2, \frac{3}{1+2p}\right) \sum_{w \in V_t} \Pr[D_{uv}^t | A_w^t]$$

Thus, we have shown that $E[ALG^t | V_t] \leq \max\left(2, \frac{3}{1+2p}\right) E[OPT^t | V_t]$. By Lemma 4.4, the claim follows. \blacktriangleleft

► Lemma 4.6. *The expected number of queries made by RANDOMQUERYPIVOT is at most $\max(4p, 1) C_{OPT}$.*

Proof. We follow an approach similar to that taken in the proof of Lemma 4.5. We will bound the number of queries made by the algorithm in each iteration t by charging queries to edges on which OPT makes a mistake and on which the algorithm makes a mistake in iteration t . Let U^t be the number of queries made by the algorithm in iteration t . We charge queries as follows to an edge $\{u, v\}$ on which OPT makes a mistake:

1. When u or v is the pivot, the algorithm makes at most 1 query on $\{u, v\}$ itself.
2. When u or v is the pivot (suppose WLOG u is the pivot), $\forall w \in T_1^{uv}$ (defined in the proof of Lemma 4.5), the algorithm makes a query on $\{u, w\}$ with probability p if $\{u, w\}$ is a $+$ edge.
3. When the pivot w is in T_1^{uv} , then with probability p at most 2 queries are made when the algorithm considers the triangle $\{u, v, w\}$.
4. Note that we need not worry about charging mistakes in $(+, +, -)$ triangles in which OPT makes 2 mistakes because when considering such a triangle the algorithm is guaranteed not to query the $-$ edge on which OPT does not make a mistake. We also need not worry about charging mistakes in $(+, +, -)$ triangles in which OPT makes 3 mistakes because each edge can be charged for any query made on that edge.

$$\begin{aligned}
 E[U^t|V_t] &\leq \sum_{\{u,v\} \subseteq E \cap (V_t \times V_t)} \frac{c_{uv}^*}{|V_t|} \left[2(1 + p|T_1^{uv}|) + \sum_{w \in T_1^{uv}} 2p \right] \\
 &\leq \sum_{\{u,v\} \subseteq E \cap (V_t \times V_t)} \frac{c_{uv}^*}{|V_t|} [2 + 4p|T_1^{uv}|]
 \end{aligned}$$

Recall from the proof of Lemma 4.5 that

$$E[OPT^t|V_t] = \sum_{\{u,v\} \subseteq E \cap (V_t \times V_t)} \frac{c_{uv}^*}{|V_t|} \sum_{w \in V_t} \Pr[D_{uv}^t|A_w^t] \geq \sum_{\{u,v\} \subseteq E \cap (V_t \times V_t)} \frac{c_{uv}^*}{|V_t|} (2 + |T_1^{uv}|)$$

Here the second inequality follows from computing $\sum_{w \in V_t} \Pr[D_{uv}^t|A_w^t]$ (see Case 1 and 7). Clearly, $\frac{2+4p|T_1^{uv}|}{2+|T_1^{uv}|} \leq \max(4p, 1)$, so $E[U^t|V_t] \leq \max(4p, 1) E[OPT^t|V_t]$. Then by Lemma 4.4, the claim follows. ◀

Theorem 4.1 follows directly from Lemmas 4.5 and 4.6.

5 Lower bound on Query Complexity

The query complexities of the algorithms presented in this paper are linear in C_{OPT} , but it is not clear whether this number of queries is necessary for finding an (approximately) optimal solution. In this section, we show that a query complexity linear in C_{OPT} is necessary for approximation factors below a certain threshold assuming that the Gap-ETH, stated below, is true.

► **Hypothesis 5.1 (Gap-ETH).** *There is some absolute constant $\gamma > 0$ s.t. any algorithm that can distinguish between the following two cases for any given 3-SAT instance with n variables and m clauses must take time at least $2^{\Omega(m)}$. (see e.g. [13])*

- i. *The instance is satisfiable.*
- ii. *Fewer than $(1 - \gamma)m$ of the clauses are satisfiable.*

The proof of the following lemma is provided in the appendix, which appears in the extended version of the paper.

► **Lemma 5.2.** *Let C_{OPT} be the optimum number of mistakes for a given instance of correlation clustering. Assuming Hypothesis 1, there is no $(1 + \frac{\gamma}{10})$ -approximation algorithm for correlation clustering on N vertices that runs in time $2^{o(C_{OPT})} \text{poly}(N)$ where γ is as defined in Hypothesis 1.*

As a corollary to the above lemma, we obtain the following.

► **Theorem 5.3.** *There is no polynomial-time $(1 + \frac{\gamma}{10})$ -approximation algorithm for correlation clustering that uses $o(C_{OPT})$ queries.*

Proof. Suppose there exists an algorithm that approximates correlation clustering with an approximation factor of $(1 + \frac{\gamma}{10})$ and uses at most $o(C_{OPT})$ queries. We follow the algorithm but instead when the algorithm issues a query, we branch to two parallel solutions instances with the two possible query answers from the oracle. Since the number of queries is $o(C_{OPT})$, the number of branches/solutions that we obtain by this process is at most $2^{o(C_{OPT})}$. We return the one which gives the minimum number of mistakes. This gives a contradiction to Lemma 5.2. ◀

6 Experiments

In this section, we report detailed experimental results on multiple synthetic and real-world datasets. We compare the performance of the existing correlation clustering algorithms that do not issue any queries, alongside with our new algorithms. We compare three existing algorithms: the deterministic constant factor approximation algorithm of Bansal et al. [7] (BBC), the combinatorial 3-approximation algorithm of Ailon et al. [1] (ACN), and the state-of-the-art 2.06-approximation algorithm of Chawla et al. based on linear program (LP) rounding [10] (LP-Rounding). The code and data used in our experiments can be found at <https://github.com/sanjayss34/corr-clust-query-esa2019>.

6.1 Datasets

Our datasets range from small synthetic datasets to large real datasets and real crowd answers obtained using Amazon Mechanical Turk. Below we give a short description of them.

Synthetics Datasets: Small

We generate graphs with ≈ 100 nodes by varying the cluster size distribution as follows. [N] represents 10 cliques whose sizes are drawn i.i.d. from a $Normal(8, 2)$ distribution. This generates clusters of nearly equal size. [S] represents 5 clusters of size 5 each, 4 clusters of size 15 each and one cluster of size 30. This generates clusters with moderate skew. [D] represents 3 cliques whose total size is 100 and whose individual sizes are determined by a draw from a $Dirichlet((3, 1, 1))$ distribution. This generates clusters with extreme skewed distribution with one cluster accounting for more than 80% of edges.

Synthetics Datasets: Large

We generate two datasets *skew* and *sqrtn* each containing 900 nodes of fictitious hospital patients data, including name, phone number, birth date and address using the data set generator of the Febrl system [11]. *skew* contains few ($\approx \log n$) clusters of large size ($\approx \frac{n}{\log n}$), moderate number of clusters ($\approx \sqrt{n}$) of moderate size ($\approx \sqrt{n}$) and a large tail of small clusters. *sqrtn* contains \sqrt{n} clusters of size \sqrt{n} .

Noise Models for Synthetic Datasets

Initially, all intra-cluster edges are labelled with + sign and all inter-cluster edges are labelled with - sign. Next, the signs of a subset of edges are flipped according to the following distributions. Denote by C_1, C_2, \dots, C_k the clusters that we generate. Let N denote the number of vertices in a graph. Let $\ell_1 = 0.01$, $\ell_2 = 0.1$, and L be an integer. For the small datasets, we set $L = 100$, and for the large *skew* and *sqrtn* datasets, we set $L = \lfloor \ell_2 \binom{N}{2} \rfloor$.

- I. Flip sign of L edges uniformly at random.
- II. Flip sign of $\min\{\lfloor L/k \rfloor, |C_i| - 1\}$ edges uniformly at random within each clique C_i . Do not flip sign of the inter-cluster edges.
- III. Flip sign of edges as in II in addition to selecting uniformly at random $\lfloor \ell_1 |C_i| |C_j| \rfloor$ edges between each pair of cliques C_i, C_j and flipping their sign.

Real-World Datasets

We use several real-world datasets.

- In the *cora* dataset [20], each node is a scientific paper represented by a string determined by its title, authors, venue, and date; edge weights between nodes are computed using Jaro string similarity [14, 25]. The *cora* dataset consists of 1.9K nodes, 191 clusters with the largest cluster-size being 236.
- In the *gym* dataset [22], each node corresponds to an image of a gymnast, and each edge weight reflects the similarity of the two images (i.e. whether the two images correspond to the same person). The *gym* dataset consists of 94 nodes with 12 clusters and maximum cluster size is 15.
- In the *landmarks* dataset [17], each node corresponds to an image of a landmark in Paris or Barcelona, and the edge weights reflect the similarity of the two images. The *landmarks* dataset consists of 266 nodes, 13 clusters and the maximum size of clusters is 43.
- In the *allsports* dataset [23], the nodes correspond to images of athletes in one of several sports, and the edge weights reflect the similarity of the two images. The pairs of images across sports are easy to distinguish but the images within the same category of sport are quite difficult to distinguish due to various angles of the body, face and uniform. The *allsports* dataset consists of 200 nodes with 64 clusters and with a maximum size of cluster being just 5.

Since the underlying graphs are weighted, we convert the edge weights to ± 1 labels by simply labeling an edge $+$ if its weight is at least $1/2$ and $-$ otherwise (the edge weights in all of the weighted graphs are in $[0, 1]$). We also perform experiments directly on the weighted graphs [10] to show how the above rounding affects the results.

Oracle

For small datasets, we use the Gurobi (www.gurobi.org) optimizer to solve the integer linear program (ILP) for correlation clustering [10] to obtain the optimum solution, which is then used as an oracle. For larger datasets like *skew*, *sqrtn* and *cora*, ILP takes prohibitively long time to run. For these large datasets, the ground-truth clustering is available and is used as the oracle.

For practical implementation of oracles, one can use the available crowd-sourcing platforms such as the Amazon Mechanical Turk. It is possible that such an oracle may not always give correct answer. We also use such crowd-sourced oracle for experiments on real datasets. Each question is asked 3 to 5 times to Amazon Mechanical Turk, and a majority vote is taken to resolve any conflict among the answers. We emphasize that the same-cluster query setting can be useful in practice because two different sources of information can produce the edge signs and the oracle – for instance, the edge signs can be produced by a cheap, automated computational method (e.g. classifiers), while the oracle answers can be provided by humans through the crowd-sourcing mechanism explained above.

6.2 Results

We compare the results of our QUERYPIVOT and RANDOMQUERYPIVOT algorithm as well as the prior algorithms BBC [7], ACN [1] and LP-Rounding [10]. For the algorithms that are randomized (ACN, LP-Rounding and RANDOMQUERYPIVOT), we report the average of three runs. The algorithm of Bansal et al. [7] requires setting a parameter δ . We tried several values of δ on several of the datasets and chose the value that seemed to give the best performance overall.

81:14 Same-Cluster Queries Bounded by Optimal Cost

■ **Table 1** Results for Experiments on synthetic small datasets. BBC denotes the algorithm of [7], ACN denotes the 3-approximation algorithm of [1], LP Rounding denotes the algorithm of [10], QP denotes QUERYPIVOT, and RQP denotes RANDOMQUERYPIVOT(0.25). All numerical columns except those marked as “Queries” give the number of mistakes made by the algorithm.

Mode	ILP Oracle	BBC	ACN	LP Rounding	QP	QP Queries	RQP	RQP Queries
N+I	100	271	205.67	100	100	113	104.33	63.3
N+II	48	104	70.0	48	48	47	56.33	27.33
N+III	93	201	130	123	93	91	97.67	66.0
D+I	100	100	267.0	100	100	86	100	83.33
D+II	48	48	144.33	48	48	57	48	40.67
D+III	64	64	216.33	64	64	71	64	75.0
S+I	100	969	206.0	100	100	136	100.67	92.0
S+II	60	831	100.67	61.67	60	71	63.67	58.67
S+III	137	913	297	141.33	137	159	139.33	107.33

■ **Table 2** Results for Large Synthetic Datasets where Mistakes are measured with respect to ground-truth clustering and the oracle is the ground-truth clustering.

Dataset/Mode	LP Rounding	BBC	ACN	QP	QP Queries	RQP	RQP Queries
Skew (I)		8175	31197.67	0	17108	71.33	10051.33
Skew (II)		700	1182.33	60	668	282	558.0
Skew (III)		8175	12260.67	56	8977	293.0	4475.67
Sqrtn (I)		13050	36251.33	0	13171	9.67	7851.0
Sqrtn (II)		0	1484.67	0	748	0.0	711.0
Sqrtn (III)		13050	12711.33	0	6449	0.0	2693.33

Synthetic Datasets

Table 1 summarizes the results of different algorithms on small synthetic datasets.

As we observe, our QUERYPIVOT algorithm always obtains the optimum clustering. Moreover, RANDOMQUERYPIVOT has a performance very close to QUERYPIVOT but often requires much less queries. Interestingly, the LP-rounding algorithm performs very well except for $N + III$. ACN and BBC algorithms have worse performance than LP-Rounding, and in most cases ACN is preferred over BBC.

For the larger synthetic datasets *skew* and *sqrtn*, as discussed the ground-truth clustering is used as an oracle. We also use the ground-truth clustering to count the number of mistakes. On these datasets, the LP-rounding algorithm caused an out-of-memory error on a machine with 256 GB main memory that we used. The linear programming formulation for correlation clustering has $O(n^3)$ triangle inequality constraints; this results in very high time and space complexity rendering the LP-rounding impractical for correlation clustering on large datasets. Table 2 summarizes the results.

As we observe, QUERYPIVOT algorithm recovers the exact ground-truth clustering in several cases. RANDOMQUERYPIVOT has a low error rate as well and uses significantly fewer queries.

■ **Table 3** Results for Real-World Datasets where mistakes are measured with respect to ground-truth clustering and the oracle is the ground-truth clustering. LP Rounding (weighted) refers to the LP rounding of [10] applied to the weighted input graph.

Dataset/ Mode	LP Rounding	LP Rounding (weighted)	BBC	ACN	QP	QP Queries	RQP	RQP Queries
Cora			62891	26065.0	4526	2188	4664.67	1474.33
Gym	221.0	332.67	449	301.67	8	150	82.67	97.33
Landmarks	29648.0	25790.0	31507	28770	3426	953	1124.67	1467.0
Allsports	230.0	226.33	227	253.33	217	41	223.67	21.0

■ **Table 4** Running times (in seconds) for the results in Table 3. For randomized algorithms, the time shown is the average over three trials.

Dataset/Mode	LP Rounding	LP Rounding (weighted)	BBC	ACN	QP	RQP
Cora			1.58	0.16	2170.33	515.59
Gym	4.96	5.09	0.004	0.0014	0.33	0.27
Landmarks	190.96	9571.32	0.048	0.00067	1.96	2.82
Allsports	41.54	42.08	0.018	0.025	13.28	12.76

Real-World Datasets

The results for the real-world datasets are reported in Table 3, 5 and 6. It is evident from Table 3 that our algorithms outperform the existing algorithms by a big margin in recovering the original clusters. Table 3 also includes results for the LP-rounding algorithm applied to the original weighted graph for the Gym, Landmarks, and Allsports datasets. We also report in Table 4 the running times for the experiments in Table 3. These numbers show that the BBC and ACN algorithms are substantially faster than the others, while our algorithms are substantially faster than the LP-rounding algorithm.

Table 5 reports the results using a faulty crowd oracle. Contrasting the results of Table 3 and 5, we observe minimal performance degradation; that is, our algorithms are robust to noise. The results in this table are important, as this setting is closest to the typical real-world application of same-cluster queries. Note that the source of information that gives the signs of the edges is different from that which is the crowd oracle. For the landmarks dataset, the original edge weights are determined by a gist detector [21], while the oracle used in Table 5 is given by high-quality crowd workers. For the gym and allsports datasets, the original edge weights are determined by (lower quality) human crowd workers, but the oracle used in Table 5 is based on high-quality crowd workers. Finally, in Table 6, we report

■ **Table 5** Results for Real-World Datasets where mistakes are measured with respect to ground-truth clustering and the oracle is the crowd.

Dataset/Mode	QP	QP Queries	RQP	RQP Queries
Gym	135	175	160.0	104.67
Landmarks	4645	1997	2172.33	1548.33
Allsports	218	41	223.67	21.0

81:16 Same-Cluster Queries Bounded by Optimal Cost

■ **Table 6** Results for Real-World Datasets where mistakes are measured with respect to the graph and the oracle is the optimal ILP solution for the graph.

Dataset/Mode	LP Rounding	BBC	ACN	QP	QP Queries	RQP	RQP Queries
Gym	276.0	464	338.0	207	171	211.0	112.67
Landmarks	4092.0	4995	5240.67	4092	267	4092.0	265.33
Allsports	33.33	65	40.67	28	36	30.33	18.67

the results using the optimum ILP solution as the oracle. For the larger datasets, it is neither possible to run the ILP nor LP-Rounding due to their huge space and time requirements. Again our algorithms perform the best, followed by the LP-rounding algorithm.

References

- 1 N. Ailon, M. Charikar, and A. Newman. Aggregating Inconsistent Information: Ranking and Clustering. *Symposium on the Theory of Computing (STOC)*, 2005.
- 2 Nir Ailon, Anup Bhattacharya, and Ragesh Jaiswal. Approximate Correlation Clustering Using Same-Cluster Queries. In *Latin American Symposium on Theoretical Informatics*, pages 14–27. Springer, 2018.
- 3 Nir Ailon, Anup Bhattacharya, Ragesh Jaiswal, and Amit Kumar. Approximate clustering with same-cluster queries. *arXiv preprint*, 2017. [arXiv:1704.01862](https://arxiv.org/abs/1704.01862).
- 4 Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):23, 2008.
- 5 H. Ashtiani, S. Kushagra, and S. Ben-David. Clustering with Same-Cluster Queries. *Advances in Neural Information Processing Systems (NIPS)*, 2016.
- 6 M. F. Balcan and A. Blum. Clustering with Interactive Feedback. *International Conference on Algorithmic Learning Theory (ALT)*, 2008.
- 7 N. Bansal, A. Blum, and S. Chawla. Correlation Clustering. *Symposium on Foundations of Computer Science (FOCS)*, 2002.
- 8 Sebastian Böcker, Sebastian Briesemeister, Quang Bao Anh Bui, and Anke Truß. Going weighted: Parameterized algorithms for cluster editing. *Theoretical Computer Science*, 410(52):5467–5480, 2009.
- 9 Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71(3):360–383, 2005.
- 10 S. Chawla, K. Makarychev, T. Schramm, and G. Yaroslavtsev. Near Optimal LP Rounding Algorithm for Correlation Clustering on Complete and Complete k-partite Graphs. *Symposium on the Theory of Computing (STOC)*, pages 219–228, 2015.
- 11 Peter Christen. Febrl: a freely available record linkage system with a graphical user interface. In *Proceedings of the second Australasian workshop on Health data and knowledge management-Volume 80*, pages 17–25. Australian Computer Society, Inc., 2008.
- 12 Erik D Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2-3):172–187, 2006.
- 13 I. Dinur. Mildly exponential reduction from Gap 3SAT to polynomial-gap label-cover. *Electronic Colloquium on Computational Complexity (ECCC)*, 2016.
- 14 Donatella Firmani, Sainyam Galhotra, Barna Saha, and Divesh Srivastava. Robust Entity Resolution Using a CrowdOracle. *IEEE Data Eng. Bull.*, 41(2):91–103, 2018.
- 15 Buddhima Gamlath, Sangxia Huang, and Ola Svensson. Semi-Supervised Algorithms for Approximately Optimal and Accurate Clustering. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 57:1–57:14, 2018. [doi:10.4230/LIPIcs.ICALP.2018.57](https://doi.org/10.4230/LIPIcs.ICALP.2018.57).

- 16 I. Giotis and V. Guruswami. Correlation Clustering with a Fixed Number of Clusters. *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2006.
- 17 Anja Gruenheid, Besmira Nushi, Tim Kraska, Wolfgang Gatterbauer, and Donald Kossmann. Fault-tolerant entity resolution with the crowd. *arXiv preprint*, 2015. [arXiv:1512.00537](https://arxiv.org/abs/1512.00537).
- 18 Shrinu Kushagra, Shai Ben-David, and Ihab Ilyas. Semi-supervised clustering for de-duplication. *arXiv preprint*, 2018. [arXiv:1810.04361](https://arxiv.org/abs/1810.04361).
- 19 A. Mazumdar and B. Saha. Clustering with Noisy Queries. *Advances in Neural Information Processing Systems (NIPS)*, 2017.
- 20 Andrew McCallum. Data. URL: <https://people.cs.umass.edu/~mccallum/data.html>.
- 21 Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision*, 42(3):145–175, 2001.
- 22 Vasilis Verroios and Hector Garcia-Molina. Entity resolution with crowd errors. In *2015 IEEE 31st International Conference on Data Engineering*, pages 219–230. IEEE, 2015.
- 23 Vasilis Verroios, Hector Garcia-Molina, and Yannis Papakonstantinou. Waldo: An adaptive human interface for crowd entity resolution. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1133–1148. ACM, 2017.
- 24 David Williams. *Probability with martingales*. Cambridge university press, 1991.
- 25 William E Winkler. Overview of record linkage and current research directions. In *Bureau of the Census*. Citeseer, 2006.

Graph Balancing with Orientation Costs

Roy Schwartz

Technion – Israel Institute of Technology, Haifa, Israel
schwartz@cs.technion.ac.il

Ran Yehekel

Technion – Israel Institute of Technology, Haifa, Israel
ran.yeheskel1@gmail.com

Abstract

Motivated by the classic GENERALIZED ASSIGNMENT PROBLEM, we consider the GRAPH BALANCING problem in the presence of orientation costs: given an undirected multi-graph $G = (V, E)$ equipped with edge weights and orientation costs on the edges, the goal is to find an orientation of the edges that minimizes both the maximum weight of edges oriented toward any vertex (makespan) and total orientation cost. We present a general framework for minimizing makespan in the presence of costs that allows us to: (1) achieve bicriteria approximations for the GRAPH BALANCING problem that capture known previous results (Shmoys-Tardos [Math. Progrm. ‘93], Ebenlendr-Krcál-Sgall [Algorithmica ‘14], and Wang-Sitters [Inf. Process. Lett. ‘16]); and (2) achieve bicriteria approximations for extensions of the GRAPH BALANCING problem that admit hyperedges and unrelated weights. Our framework is based on a remarkably simple rounding of a strengthened linear relaxation. We complement the above by presenting bicriteria lower bounds with respect to the linear programming relaxations we use that show that a loss in the total orientation cost is required if one aims for an approximation better than 2 in the makespan.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms; Mathematics of computing → Approximation algorithms

Keywords and phrases Graph Balancing, Generalized Assignment Problem

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.82

Funding *Roy Schwartz*: Supported by ISF grant 1336/16 and NSF-BSF grant number 2016742.

1 Introduction

We consider the GRAPH BALANCING problem (GB) where we are given an undirected multi-graph $G = (V, E)$ equipped with edge weights $p : E \rightarrow \mathbb{R}^+$. The goal is to orient all the edges of the graph, where each edge can be oriented to one of its endpoints. Given an orientation of the edges the load of a vertex u is the sum of weights of edges oriented toward it. The goal is to find an orientation of the edges that minimizes the maximum load over all vertices.

GB was first introduced by Ebenlendr et al. [3] and since its introduction it has attracted much attention (see, e.g., [10, 18, 6, 2, 12]). Besides being a natural graph optimization problem on its own, a main motivation for considering GB is the well known GENERALIZED ASSIGNMENT PROBLEM (GAP) (see, e.g., [15, 3, 16, 19]). In GAP we are given a collection \mathcal{M} of m machines and a collection \mathcal{J} of n jobs, along with processing times $p_{i,j}$ (the processing time of job j on machine i) and assignment costs $c_{i,j}$ (the cost of assigning job j to machine i). Each job must be assigned to one of the machines. The processing time of machine i is the sum of processing times $p_{i,j}$ over all jobs j that are assigned to i , and the makespan of an assignment is the maximum over all machines i of its processing time. Additionally, the total assignment cost of an assignment is the sum of assignment costs $c_{i,j}$ over all machines i and jobs j that are assigned to i . Given a target makespan T , we denote by $C(T)$ the minimum total assignment cost over all assignments with makespan at most T . If there are no assignments with makespan at most T , then $C(T) = \infty$. The goal in GAP, given a



© Roy Schwartz and Ran Yehekel;
licensed under Creative Commons License CC-BY
27th Annual European Symposium on Algorithms (ESA 2019).

Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 82; pp. 82:1–82:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

target makespan T , is to find an assignment with makespan at most T and total assignment cost at most $C(T)$, or declare that no such assignment exists. We note that only T is given to the algorithm whereas $C(T)$ is not. For this bicriteria problem, the celebrated result of Shmoys and Tardos [15] provides an approximation algorithm that finds an assignment with makespan at most $2T$ and total assignment cost at most $C(T)$.

GB is captured by GAP since one can: (1) set \mathcal{M} to be V and \mathcal{J} to be E ; and (2) for each job $j \in \mathcal{J}$ (which corresponds to an edge $e \in E$) set its processing time to be p_e for the two machines that correspond to the endpoints of e and ∞ for all other machines. Note that assigning job j to machine i corresponds to orienting the edge e toward its endpoint that corresponds to machine i . There are two important things to note. First, GB was originally defined as a *single criterion* optimization problem as opposed to GAP which is a bicriteria optimization problem. Second, the weights p in GB, which represent the processing times of the jobs, are *related*, *i.e.*, the processing times do not depend on the vertex the edge is oriented to. Ebenlendr et al. [3] introduced a novel linear relaxation and rounding algorithm that achieves an approximation of 1.75 with respect to the optimal makespan. They also proved that even for this special case, no polynomial time algorithm can achieve an approximation less than 1.5 unless $P = NP$, thus extending the hardness of GAP to GB.

In this work we consider the *bicriteria* GB problem, where we are also given orientation costs, the equivalent to the assignment costs in GAP. Formally, an edge $e = (u, v)$ has orientation costs $c_{e,u}$ and $c_{e,v}$ and orienting it to u incurs a cost of $c_{e,u}$. Similarly to GAP, given a target makespan T , the goal is to find an orientation of the edges with total orientation cost at most $C(T)$ and makespan at most T .¹ To the best of our knowledge, the bicriteria GB problem was not previously considered. We say that an algorithm is a (α, β) -approximation if given a target makespan T , it outputs an orientation with makespan at most αT and total orientation cost at most $\beta C(T)$. Thus, [15] is a $(2, 1)$ -approximation to GB. We note that the algorithm of [3] cannot handle orientation costs and is in fact a $(1.75, \infty)$ -approximation for GB. A result by Wang and Sitters [18] implicitly gives a $(11/6, 3/2)$ -approximation for GB.

We study the bicriteria tradeoff between makespan and total orientation cost in GB, presenting both upper and lower bounds (the latter are with respect to the linear programming relaxations used in this work). We employ a remarkably simple general framework that allows us to achieve bicriteria approximations for GB that capture and extend known results. Furthermore, we consider extensions of GB that allow for: (1) hyperedges to be present, *i.e.*, a job can be assigned to more than two machines; and (2) processing times can be unrelated, *i.e.*, the processing time of a job might depend on the machine it is assigned to. Our results regarding these extensions improve upon the previously best known algorithms, and are also based on the general framework presented in this paper. We believe this framework might be of independent interest to other related scheduling problems.

1.1 Our Results

Our results are of three different flavors: bicriteria upper bounds for GB, bicriteria lower bounds for GB, and both upper and lower bicriteria bounds for extensions of GB (all lower bounds are with respect to the linear programming relaxations we use). Let us now elaborate on each of the above.

¹ As in GAP, the total orientation cost of an orientation is defined as the sum of orientation costs $c_{e,u}$ over all vertices u and edges e oriented toward u . $C(T)$ is defined as the minimum total orientation cost over all orientations with makespan at most T . If no such orientation exists then $C(T)$ is set to ∞ .

1.1.1 Upper Bounds

We present a general framework for minimizing makespan in the presence of costs and obtain two algorithms that achieve bicriteria approximations for GB. This is summarized in the following two theorems.

► **Theorem 1.** *There exists a polynomial time algorithm that finds an orientation that is a $(1.75 + \gamma, 1/(2\gamma+0.5))$ -approximation for GRAPH BALANCING, for every $1/12 - \epsilon \leq \gamma \leq 1/4$ where $\epsilon = \sqrt{33}/4 - 17/12 \approx 0.02$.*

► **Theorem 2.** *There exists a polynomial time algorithm that finds an orientation that is a $(1.75 + \gamma, 1 + 1/\gamma)$ -approximation for GRAPH BALANCING, for every $0 \leq \gamma \leq 1/4$.*

Both the above theorems provide a smooth tradeoff between makespan and orientation cost while capturing previous known results for GB as special cases, *i.e.*, Theorem 1 captures the $(2, 1)$ and $(11/6, 3/2)$ approximations of [15] and [18] for $\gamma = 1/4$ and $\gamma = 1/12$ respectively, whereas Theorem 2 captures the $(1.75, \infty)$ -approximation of [3] for $\gamma = 0$. Theorem 1 is depicted in Figure 1.

1.1.2 Lower Bounds

We present bicriteria lower bounds for GB. As previously mentioned, our lower bounds apply to a strengthening of the relaxation of [3], which we denote by LP_k (see subsection 3.2). The lower bound is summarized in the following theorem and is depicted in Figure 1.

► **Theorem 3.** *For every $0 \leq \gamma < 1/4$ and $\epsilon > 0$, there exists an instance for GRAPH BALANCING and target makespan T such that: (1) LP_k is feasible and has value of OPT_{LP_k} , and (2) every orientation whose makespan is at most $(1.75 + \gamma)T$ has orientation cost of at least $1/(\gamma+0.75+\epsilon)OPT_{LP_k}$.*

To the best of our knowledge, all algorithms for GB that find an orientation that achieves an approximation better than 2 with respect to the makespan use the relaxation of [3] (or no relaxation at all, *e.g.*, [6]).²

1.1.3 Extensions

Using our general framework, we present bicriteria algorithms for extensions of GB. The extensions of GB we consider allow hyperedges and unrelated weights to the edges. It is important to note that all the upper bounds presented below hold for the single criterion versions of these problems as well. In particular, we achieve an approximation strictly better than 2, with respect to the makespan, to several problems that capture GB and are not captured by the RESTRICTED ASSIGNMENT problem (RA).³ To the best of our knowledge, this is the first polynomial time algorithm with approximation factor better than 2 to the makespan for problems that capture GB and are not captured by RA. Let us now elaborate on these extensions.

² Recently, Jansen and Rohwedder [10] showed that using a different stronger relaxation called the configuration LP one can achieve an approximation of less than 1.75 to the makespan. However, this result does not produce a polynomial time algorithm that orients the edges but rather only approximates the *value* of the optimal makespan. Moreover, this result has an unbounded loss with respect to the orientation cost.

³ The RA is a special case of GAP where each job has a set of machines it can be assigned to, and has an equal processing time on each of them.

The first extension allows for light unrelated hyperedges. Formally, given $\beta \in [0, 1]$, the input can contain hyperedges whose weight with respect to the vertices it shares may vary, as long as it does not exceed β (we may assume without loss of generality that the largest weight in p equals 1). We denote this problem by GRAPH BALANCING WITH UNRELATED LIGHT HYPER EDGES (GBUH(β)). A special case of this problem was introduced by Huang and Ott in [6] who presented a $(\frac{5}{3} + \beta/3, \infty)$ -approximation when $\beta \in [4/7, 1)$. We improve upon [6] in three aspects. First, we consider the general bicriteria problem, *i.e.*, orientation costs are present, and achieve bounded loss with respect to the total orientation cost (recall that [6] cannot handle orientation costs). Second, we allow any $\beta \in [0, 1]$, where [6] allows for $\beta \in [4/7, 1)$ only. Third, we allow the hyperedges to be unrelated, *i.e.*, different weights to different endpoints, where hyperedge weights in [6] are related. Our result for this extension is summarized in the following theorem.

► **Theorem 4.** *Let $0 \leq \beta \leq 1$. For every $\max\{1/12, \beta/3 - 1/12\} \leq \gamma \leq 1/4$, there exists a polynomial time algorithm that finds an orientation that is a $(1.75 + \gamma, 1/(2\gamma+0.5))$ -approximation to GBUH(β).*

The second extension further generalizes the first one, and it also allows edges to have unrelated weights as long as the weights are greater than β . Unfortunately, we prove that this problem in its full generality is as hard to approximate as GAP. However, if it is assumed that the optimal makespan is at least 1 (as before we can assume without loss of generality that the largest weight in p equals 1), we can achieve improved results. We denote this problem by GRAPH BALANCING WITH UNRELATED LIGHT HYPER EDGES AND UNRELATED HEAVY EDGES (GBU(β)).⁴

► **Theorem 5.** *Let $\beta \geq \sqrt{2} - 1$. For every $\beta/3 - 1/12 \leq \gamma \leq 1/4$, there exists a polynomial time algorithm that finds an orientation that is a $(1.75 + \gamma, 1/(2\gamma+0.5))$ -approximation to GBU(β).*

We prove that there are values of β for which the bicriteria approximation of Theorem 5 is tight. Specifically, we prove the latter for $\beta = 1/2$ and LP_k . The lower bounds are summarized in the following theorem.

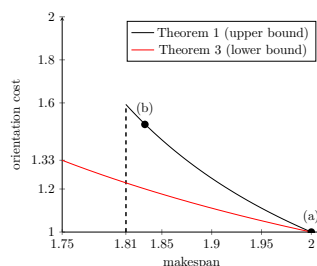
► **Theorem 6.** *For every $\epsilon > 0$, there exists an instance of GBU(0.5) that is feasible to LP_k and every orientation has a makespan of at least $11/6 - \epsilon$. Moreover, for every $1/12 \leq \gamma \leq 1/4$, target makespan T and $\epsilon > 0$, there exists an instance for GBU(0.5) that is feasible to LP_k and has a value of OPT_{LP_k} , and every orientation with makespan at most $(1.75 + \gamma)T$ has an orientation cost of at least $(1-\epsilon)/(2\gamma+0.5) \cdot OPT_{LP_k}$.*

In the third and final extension we allow the edges in GB to be unrelated, but the weights cannot vary arbitrarily. Formally, given a parameter $c \geq 1$, every edge $e = (u, v)$ satisfies $p_{e,u} \leq c \cdot p_{e,v}$ and $p_{e,v} \leq c \cdot p_{e,u}$. We denote this problem by SEMI-RELATED GRAPH BALANCING (SRGB(c)). The following theorem summarizes our algorithm for SRGB(c).

► **Theorem 7.** *There exists a polynomial time algorithm to SRGB(c), that finds an orientation that is a $(1.5 + 0.5a, 1/a)$ -approximation, where a is the root in the range $[0.5, 1]$ of the polynomial:*

$$(1/c + 1/2) \cdot a^3 + (5/(2c) - 1/2) \cdot a^2 - 7/(2c) \cdot a + 1/c.$$

⁴ While the assumption that the largest weight p equals 1 implies that the optimal makespan is least 1 for GB and GBUH(β), this is not necessarily the case when the edge weights might be unrelated. Thus, the assumption in GBU(β) that the optimal makespan is at least 1 is not without loss of generality.



■ **Figure 1** Our bicriteria bounds for GRAPH BALANCING. (a) is given in Shmoys and Tardos [15], whereas (b) is implicitly given in Wang and Sitters [18].

We remark that the approximation guaranteed by Theorem 7 is never worse than 2 since it can be proved that $a = 1 - \Omega(1/c)$, yielding a $(2 - \Omega(1/c), 1 + O(1/c))$ -approximation. It is worth noting that when $c = \infty$, which corresponds to the most general case, even the configuration LP has an integrality gap of 2 with respect to the makespan (see [4, 17]).

1.2 Our Techniques

We present a remarkably simple framework that allows us to provide bicriteria upper bounds for both GB and its extensions, *i.e.*, $\text{GBUH}(\beta)$, $\text{GBU}(\beta)$, and $\text{SRGB}(c)$. The framework is based on rounding of a strengthening of the linear relaxation of [3].

The rounding is comprised of two complementary steps, the first *local* and the second *global*. Intuitively, in the first local step, each edge can be oriented to one of its endpoints in case the relaxation indicates a strong (fractional) inclination toward that endpoint. We note that in order to quantify this inclination the weight of the edge is taken into account, where lighter edges are less likely to be oriented. Specifically, denote by $x_{e,u} \in [0, 1]$ how much the relaxation fractionally orients edge $e = (u, v)$ toward its endpoint u . The local step orients e toward u if $x_{e,u} > f(p_e)$ for some non-increasing threshold function $f : [0, 1] \rightarrow [1/2, 1]$. As previously mentioned, this step is considered *local* since only $x_{e,u}$ and p_e are used to determine whether to orient e , and if so to which of its two endpoints.⁵ In the second global step of the rounding, we consider the remaining edges which were not yet oriented in the first local step and apply the algorithm of Shmoys and Tardos [15] which finds a minimum cost perfect matching in a suitable bipartite graph. As previously mentioned, this step is considered *global* since all edges which are not yet oriented are taken into consideration when computing the matching.

The above two-phase rounding is not sufficient on its own to obtain our claimed results, and we further strengthen the relaxation of [3] by forcing additional new constraints. Intuitively, for every vertex u our constraints state that if a collection of edges S touching u has total weight of more than T then not all edges in S can be chosen. We enforce the above constraints for all subsets of size at most k , for some fixed parameter k , resulting in a strengthened linear relaxation which we denote by LP_k . It is important to note that these constraints cannot be inferred from the original relaxation of [3], and thus are required in our analysis of the above two-phase rounding.

⁵ This rounding was used in Wang and Sitters [18] with a specific “step” threshold function f to implicitly obtain a $(1^{1/6}, 3/2)$ -approximation for GB.

1.3 Additional Related Work

Lenstra et al. [11] introduced the classic well known 2-approximation to the single criterion GAP. They also proved that no polynomial time algorithm can approximate the makespan within a factor less than 1.5 unless $P = NP$. This was followed by Shmoys and Tardos [15] who introduced the bicriteria GAP and presented a $(2, 1)$ -approximation for it. A slightly improved approximation of $2 - 1/m$ for the makespan was given by Shchepin and Vakhania [14]. If the number of machines is fixed polynomial time approximation schemes are known [5, 8]. For the case of uniformly related machines (each machine i has speed s_i and assigning job j to machine i takes p_j/s_i time) Hochbaum and Shmoys [13] presented a polynomial time approximation scheme. The RESTRICTED ASSIGNMENT problem (RA) is a special case where each job has an equal processing time on the machines it can be assigned to (for every job j and machine i : $p_{i,j} \in \{p_j, \infty\}$). For this special case, Svensson [16] proved that one can approximate the value of the optimal makespan by a factor of $33/17$ using the configuration LP, that was first introduced by Bansal et al. for the SANTA CLAUS PROBLEM [1]. This was subsequently improved by Jansen and Rohwedder [9] who presented an approximation of $11/6$. If one further assumes that the processing times have only two possible values [7] presented an improved approximation of $5/3$. The above results [16, 9, 7] do not present polynomial time algorithms that produce a schedule with the promised makespan, but only approximate the value of the makespan.

When considering GB, Jansen and Rohwedder [10] recently showed a similar flavor result: using the configuration LP one can estimate the value of the optimal makespan by a factor of $1.75 - \epsilon$, for some small constant $\epsilon > 0$. However, as before, [10] does not produce an orientation in polynomial time. The special case of GB where only two processing times are present admits a (tight) 1.5-approximation (given independently by [6, 2, 12]).

To the best of our knowledge, no work on GB considered orientation costs and in particular the tradeoff between makespan and orientation cost.

1.4 Paper Organization

Section 2 contains the required preliminaries. In Section 3 we present our general framework and apply it to GB to obtain bicriteria algorithms. Section 4 contains our bicriteria lower bound for GB. Finally, in Section 5 we consider the mentioned extensions of GB and apply the framework to these extensions to obtain improved algorithms.

2 Preliminaries

Given a multi-graph $G = (V, E)$ and a vertex $u \in V$ denote by $\delta(u) \triangleq \{e \in E \mid u \in e\}$ the collection of edges incident to u . In addition define: $\mathcal{F}(u) \triangleq \{S \subseteq \delta(u) \mid \sum_{e \in S} p_e \leq 1\}$, *i.e.*, the collection of feasible subsets of edges incident to u (for simplicity of presentation we further assume without loss of generality that $T = 1$ since we can scale all processing times by T). Moreover, we denote by OPT_{LP} and OPT_{LP_k} the optimal value of a feasible solution to the relaxation LP and LP_k respectively.

The algorithm of Shmoys and Tardos [15] is a key ingredient in our framework, thus we present it not only for completion but also since understanding its inner-working helps in analyzing our algorithms. Recall that [15] is a $(2, 1)$ -approximation for GAP. We assume without loss of generality that $T = 1$ since one can scale the processing times by T . First, the relaxation in Figure 2 is solved, where \mathcal{J} is the set of jobs and \mathcal{M} is the set of machines.

$$\begin{aligned}
& (LP_{GAP}) \\
\min & \quad \sum_{j \in \mathcal{J}} \sum_{i \in \mathcal{M}} x_{i,j} c_{i,j} \\
\text{s.t.} & \quad \sum_{i \in \mathcal{M}} x_{i,j} = 1 \quad \forall j \in \mathcal{J} \quad (\text{Job}) \\
& \quad \sum_{j \in \mathcal{J}} x_{i,j} p_{i,j} \leq 1 \quad \forall i \in \mathcal{M} \quad (\text{Load}) \\
& \quad x_{i,j} = 0 \quad \forall i \in \mathcal{M}, j \in \mathcal{J} : p_{i,j} > 1 \\
& \quad x_{i,j} \geq 0 \quad \forall i \in \mathcal{M}, j \in \mathcal{J}
\end{aligned}$$

■ **Figure 2** The relaxation by Shmoys and Trados [15] to GAP.

The variable $x_{i,j}$, for each $i \in \mathcal{M}$ and $j \in \mathcal{J}$, indicates whether job j is scheduled on machine i . Note that if there is no feasible solution to the relaxation, then the algorithm declares there is no schedule with makespan at most T .

Given a solution \mathbf{x} to LP_{GAP} , the algorithm of [15] constructs a weighted bipartite graph $G = (\mathcal{J}, S, E)$, which will be described shortly. Afterwards, the algorithm finds a minimum cost perfect matching to the side \mathcal{J} , *i.e.*, each vertex in \mathcal{J} is matched to a vertex in S . Using this matching the algorithm assigns each job to a machine. The bipartite graph G is constructed as follows, where we assume that $\mathcal{J} = \{1, 2, \dots, n\}$ is the set of jobs and S is a collection of “slots”. Machine i is allocated $k_i \triangleq \lceil \sum_{j=1}^n x_{i,j} \rceil$ slots which we denote by $slot(i, 1), \dots, slot(i, k_i)$, each having a capacity of 1. For each machine i sort the jobs in a non-increasing order of their processing time $p_{i,j}$, and for each job j in this order add $x_{i,j}$ units of job j to the next non-full slot of machine i (starting from $slot(i, 1)$). If $x_{i,j}$ is larger than the remaining capacity of the slot, which we denote by r , add r units of job j to that slot and $x_{i,j} - r$ units of job j to the next slot. An edge connecting job j and a slot (i, ℓ) is added to E if some of the $x_{i,j}$ units of j were added to the slot (i, ℓ) , and its cost is set to $c_{i,j}$. A description of [15] appears in Algorithm 1.

■ **Algorithm 1** Shmoys-Tardos $(\mathbf{x}, \mathbf{p}, \mathbf{c})$.

-
- 1 Construct the bipartite graph $G = (\mathcal{J}, S, E)$ as described above.
 - 2 Find in G a minimum cost perfect matching with respect to \mathcal{J} .
 - 3 For each job $j \in \mathcal{J}$, assign j to machine i if the slot that is matched to j belongs to i .
-

We say a slot is *full* if the remaining capacity of that slot is 0. Additionally, we say a job j is on *top* of a slot if j is the first job to be inserted to that slot. It can be proved that the load on machine i in the output of Algorithm 1 is at most $1 + p_{i,1}$, where $p_{i,1}$ is the processing time of the job on top of $slot(i, 1)$, *i.e.*, the largest processing time of a job that is fractionally scheduled on machine i . Since, $p_{i,1} \leq 1$, the makespan of the assignment is at most 2. Furthermore, it can be shown that the cost of the assignment is at most $OPT_{LP_{GAP}}$, and thus at most $C(T)$.

We remark that when one is aiming to solve the single criterion version of this problem, *i.e.*, finding an assignment that minimizes the makespan, a binary search could be performed to find the smallest T such that the linear relaxation is feasible. In general, any (α, β) -approximation for the bicriteria problem implies an approximation of α for the single criteria problem.

$$\begin{aligned}
& (LP) \\
\min & \sum_{e \in E} \sum_{u \in e} x_{e,u} c_{e,u} \\
\text{s.t.} & \sum_{u \in e} x_{e,u} = 1 \quad \forall e \in E \quad (\text{Edge}) \\
& \sum_{e \in \delta(u)} x_{e,u} p_e \leq 1 \quad \forall u \in V \quad (\text{Load}) \\
& \sum_{e \in \delta(u): p_e > 0.5} x_{e,u} \leq 1 \quad \forall u \in V \quad (\text{Star}) \\
& x_{e,u} \geq 0 \quad \forall u \in V, e \in \delta(u)
\end{aligned}$$

■ **Figure 3** The relaxation by Ebenlendr et al. [3] to GB.

3 The General Framework and Graph Balancing

We start by describing the general framework in the setting of GB. For simplicity of presentation, given a target makespan T , if there exists an edge e such that $p_e > T$ the algorithm immediately declares that there is no orientation with makespan at most T . Otherwise, we scale the processing times by T . Thus, without loss of generality, $T = 1$ and $p_e \leq 1$ for every $e \in E$.

Currently, we consider the relaxation of [3], which we denote by LP , with the addition of an objective function that minimizes the orientation cost.⁶ This relaxation appears in Figure 3.

Note that the Star constraint of LP implies that at most a total fraction of 1 of *big* edges, *i.e.*, edges whose weight is larger than $1/2$, can be oriented toward u . Moreover, we note that later we strengthen this relaxation by adding additional constraints.

Once the processing times are scaled by T , the algorithm solves the relaxation LP . If there is no feasible solution to the relaxation, then the algorithm declares that there is no orientation with makespan at most T . Thus, from this point onward we assume that LP is feasible and focus on the rounding.

Recall that the rounding consists of only two steps, the first local and the second global. In the first step, some of the edges might be oriented, where an edge e is oriented toward u if $x_{e,u} > f(p_e)$ for a given threshold function $f : [0, 1] \rightarrow [1/2, 1]$. We employ the framework for threshold functions f which are monotone non-increasing, thus making lighter edges less likely to be oriented compared to heavier edges. In the second step, the remaining un-oriented edges are oriented using Algorithm 1. The framework is described in Algorithm 2. It receives as an input: (1) the graph $G = (V, E, \mathbf{p}, \mathbf{c})$; (2) \mathbf{x} a solution to the relaxation; (3) a threshold function f .

■ **Algorithm 2** Framework($G = (V, E, \mathbf{p}, \mathbf{c}), \mathbf{x}, f$).

-
- 1 For each edge e and $u \in e$: if $x_{e,u} > f(p_e)$ then orient e to u and remove e from E .
(Local Step)
 - 2 Execute Algorithm 1. (Global Step)
-

⁶ In Section 5 we also need the constraint that appears in the relaxation of [15] which states that $x_{e,u} = 0$ if $p_{e,u} > 1$, for every $e \in E$ and $u \in e$.

Note that the *Local Step* of Algorithm 2 is well defined, *i.e.*, no edge is oriented to both its endpoints. This is due to the Edge constraints and the fact that for each $p \in [0, 1]$: $f(p) \geq 1/2$. Note that the framework captures Algorithm 1 as a special case since one can choose $f \equiv 1$. We now focus on bounding the makespan and orientation cost produced by the framework, for a general threshold function f . This analysis will be useful for the rest of the paper.

Let us start by focusing on bounding the makespan. We start by presenting a simple but crucial observation. The observation states that if an edge $e = (u, v)$ was *not* oriented at the *Local Step* then $x_{e,u}$ and $x_{e,v}$ cannot vary much. It is important to note that this is the *only* place in our proof we use the fact that e is an edge, *i.e.*, the job that corresponds to e can be assigned to only two machines u and v (otherwise our algorithm could have been applied to the more general problem of RA).

► **Observation 1.** *Let $e = (u, v) \in E$ such that e was not oriented to either u or v in the *Local Step*. Then $1 - f(p_e) \leq x_{e,u} \leq f(p_e)$.*

Proof. e was not oriented toward u in the *Local Step*, and therefore $x_{e,u} \leq f(p_e)$. Additionally, the Edge constraint implies that $x_{e,v} = 1 - x_{e,u}$, and since e was not oriented toward v in the *Local Step* then $1 - x_{e,u} \leq f(p_e)$. This concludes the proof. ◀

Now we focus on bounding the makespan. Fix a vertex $u \in V$, and denote the slots that were allocated to u in Algorithm 1 by: $slot(u, 1), \dots, slot(u, k)$ or alternatively by s_1, \dots, s_k . For $i \in \{1, 2, \dots, k\}$ let e_i be the edge on top of $slot(u, i)$ and denote its processing time by p_i . We assume without loss of generality that $p_{k+1} \triangleq 0$ and $x_{e_{k+1}, u} = 1$ (one can simply add a 0 weight edge that is fully oriented toward u).⁷ Additionally, denote by e'_1, \dots, e'_t the edges that were oriented to u in the *Local Step*, and denote by q_1, \dots, q_t , their processing times respectively. Lastly, for a slot s and edge e we denote by $y_{e,s}$ the fraction that e is assigned to s .

We now introduce a new observation that lower bounds the fractional load in the first slot of u , *i.e.*, $\sum_{e \in slot(u, 1)} y_{e, s_1} p_e$. This observation will be useful in bounding the load on u .

► **Observation 2.** *The fractional load in the first slot of u is at least:*

$$\sum_{e \in slot(u, 1)} y_{e, s_1} p_e \geq (1 - f(p_1))p_1 + f(p_1)p_2.$$

Proof. From Observation 1 we know that $x_{e_1} \geq 1 - f(p_1)$. Moreover, since e_1 is the first edge to be inserted to the first slot, then it is contained fully in $slot(u, 1)$. Therefore, $y_{e_1, s_1} = x_{e_1, u} \geq 1 - f(p_1)$. Recall that $p_2 \leq p_1$. Since $slot(u, 1)$ is full and its capacity equals 1, we can conclude: $\sum_{e \in slot(u, 1)} y_{e, s_1} p_e \geq f(p_1)p_1 + (1 - f(p_1))p_2$. ◀

Now we introduce a lemma that is inspired by [15] and upper bounds the load on u .

► **Lemma 8.** *Let e'_1, \dots, e'_t be the edges that were oriented to u in the *Local Step*, and let q_1, \dots, q_t be their processing times respectively. Then,*

$$\sum_{i=1}^t q_i + \sum_{i=1}^k p_i \leq 1 + \sum_{i=1}^t (1 - f(q_i))q_i + f(p_1)p_1 + (1 - f(p_1))p_2.$$

⁷ Alternatively, we can also assume $p_{k+2} = 0$ and $x_{e_{k+2}, u} = 1$ as well.

82:10 Graph Balancing with Orientation Costs

Proof. First, recall that for every $1 \leq s \leq k-1$ $slot(u, s)$ has a capacity exactly 1. Moreover, the slots are filled with edges in decreasing order of processing time. Therefore, we can deduce that for each $1 \leq i \leq k-1$:

$$\sum_{e \in slot(u, i)} y_{e, s_i} p_e \geq \sum_{e \in slot(u, i)} y_{e, s_i} p_{i+1} = p_{i+1} \sum_{e \in slot(u, i)} y_{e, s_i} = p_{i+1}.$$

Since at most one edge from each slot can be selected in the *Global Step*, the load on u from edges that oriented to u in the *Global Step* is at most $\sum_{i=1}^k p_i$. From the above inequality, along with Observation 2, we can conclude that:

$$\begin{aligned} \sum_{i=1}^t q_i + \sum_{i=1}^k p_i &= \sum_{i=1}^t q_i + p_1 + p_2 + \sum_{i=3}^k p_i \leq \sum_{i=1}^t q_i + p_1 + p_2 + \sum_{i=2}^{k-1} \sum_{e \in slot(u, i)} y_{e, s_i} p_e \\ &\leq \sum_{i=1}^t q_i + p_1 + p_2 + \sum_{i=1}^k \sum_{e \in slot(u, i)} y_{e, s_i} p_e - \sum_{e \in slot(u, 1)} y_{e, s_1} p_e \\ &\leq \sum_{i=1}^t q_i + p_1 + p_2 + \sum_{e \in \delta(u)} x_{e, u} p_e - ((1 - f(p_1))p_1 + f(p_1)p_2) \\ &\leq \sum_{i=1}^t q_i + f(p_1)p_1 + (1 - f(p_1))p_2 + \left(1 - \sum_{i=1}^t f(q_i)q_i\right) \\ &= 1 + \sum_{i=1}^t (1 - f(q_i))q_i + f(p_1)p_1 + (1 - f(p_1))p_2. \end{aligned}$$

The last inequality follows from the Load constraint on u , and the fact that the edges e'_1, \dots, e'_t were removed from E at the end of the *Local Step*. \blacktriangleleft

Lastly, we observe that all of the *big* edges, *i.e.*, edges whose weight is larger than $1/2$, that are not oriented toward u in the *Local Step* are assigned to the first slot. This is summarized in the following observation.

► **Observation 3.** *Let e be an edge in $slot(u, i)$ such that $i > 1$. Then, $p_e \leq 1/2$.*

Proof. Assume for the sake of contradiction that $p_e > 1/2$. Since the slots are filled in a non-increasing weight order, all edges in slots $1, 2, \dots, i-1$ are filled with fractions of edges whose processing time is greater than $1/2$. Therefore, $\sum_{e \in \delta(u): p_e > 1/2} x_{e, u} > 1$, which contradicts the Star constraint on u . \blacktriangleleft

Let us now focus on the orientation cost. The following lemma upper bounds the orientation cost of the orientation produced by Algorithm 2.

► **Lemma 9.** *Given $f : [0, 1] \rightarrow [0, 1/2]$, let $c \triangleq (\inf\{f(p) | p \in [0, 1]\})^{-1}$. Then Algorithm 2 with f outputs an orientation with a cost of at most $c \cdot C(T)$.*

3.1 Graph Balancing – Upper Bound on Tradeoff Between Makespan and Orientation Cost

Let us now focus on applying the framework, with an appropriate threshold function f , to GB. First, we present a theorem that achieves part of the tradeoff claimed in Theorem 1, and only in the next subsection we show how to extend this tradeoff to fully achieve Theorem 1.

► **Theorem 10.** *There exists a threshold function f such that Algorithm 2 finds an orientation that is a $(1.75 + \gamma, 1/(2\gamma+0.5))$ -approximation, for every $1/12 \leq \gamma \leq 1/4$.*

The function f_α we use in the proof of Theorem 10 is the following:

$$f_\alpha(p_e) = \begin{cases} 1 & \text{if } p_e \leq 1/2 \\ \alpha & \text{if } p_e > 1/2 \end{cases} \quad (1)$$

where $2/3 \leq \alpha \leq 1$. The following lemma upper bounds the makespan of Algorithm 2 with the above f_α .

► **Lemma 11.** *The makespan of the orientation produced by Algorithm 2 with f_α is at most $1.5 + 0.5\alpha$, where $2/3 \leq \alpha \leq 1$.*

Proof. Consider the number of edges that were oriented toward u in the *Local Step*. First, we note that from the Star constraint on u , at most one edge can be oriented toward u in the *Local Step*. If this is not the case then let e'_1 and e'_2 be edges oriented to u in the *Local Step*. Then, $p_{e_1}, p_{e_2} > 1/2$. However, $x_{e_1,u} + x_{e_2,u} > \alpha + \alpha \geq 2/3 + 2/3 > 1$, which contradicts the Star constraint on u . Hence, there are only two cases to consider.

Case 1: Assume no edge is oriented toward u in the *Local Step*. Therefore, using Lemma 8 and Observation 3 the load on u is at most:

$$\begin{aligned} \sum_{i=1}^k p_i &\leq 1 + f_\alpha(p_1)p_1 + (1 - f_\alpha(p_1))p_2 \leq 1.5 + f_\alpha(p_1)(p_1 - 0.5) \\ &\leq 1.5 + \alpha \cdot (1 - 0.5) = 1.5 + 0.5\alpha, \end{aligned}$$

where the last inequality follows from the fact that the expression: $f_\alpha(p_1)(p_1 - 0.5)$ is maximized when $p_1 = 1$ (and thus $f_\alpha(p_1) = \alpha$).

Case 2: Assume there is exactly one edge that was oriented toward u in the *Local Step*. Recall we denote this edge as e'_1 and its processing time by q_1 . Since $q_1 > 1/2$ and $x_{e'_1,u} > \alpha$, then it must be the case that $p_1 \leq 1/2$ (otherwise Observation 1 implies that $x_{e'_1,u} + x_{e_1,u} > \alpha + 1 - \alpha = 1$, which contradicts the Star constraint for u). Therefore, from Lemma 8 the load on u in the output of Algorithm 2 is at most:

$$\begin{aligned} q_1 + \sum_{i=1}^k p_i &\leq 1 + (1 - f_\alpha(q_1))q_1 + f_\alpha(p_1)p_1 + (1 - f_\alpha(p_1))p_2 \leq 1 + (1 - \alpha)q_1 + p_1 \\ &\leq 1 + (1 - \alpha) + 0.5 = 2.5 - \alpha \leq 1.5 + 0.5\alpha. \end{aligned}$$

The second inequality follows from the fact that $p_2 \leq p_1$ and $f_\alpha(q_1) = \alpha$ (since $q_1 > 1/2$), whereas the third inequality from the fact that $p_1 \leq 1/2$. In addition, the last inequality follows from the fact that $2/3 \leq \alpha \leq 1$. ◀

Now, we are ready to conclude the proof of Theorem 10:

Proof of Theorem 10. Applying Lemma 11, Lemma 9 and choosing $\gamma = 0.5\alpha - 0.25$ finishes the proof. ◀

We now show that the analysis of Algorithm 2 with a threshold function f_α is tight. Formally, we prove the following lemma:

82:12 Graph Balancing with Orientation Costs

► **Lemma 12.** *For every $1/2 \leq \alpha < 1$ there exists an instance such that the output of Algorithm 2 with f_α has makespan at least $\max\{1.5 + 0.5\alpha, 2.5 - \alpha\}$ and orientation cost at least $1/\alpha \cdot OPT_{LP}$.*

Lemma 12 shows the analysis of Algorithm 2 with a threshold function f_α is tight. Consequently, in order to extend the bicriteria tradeoff of Theorem 10, and obtain Theorem 1, we require a different threshold function and a stronger relaxation.

3.2 Graph Balancing – Extending the Tradeoff

It is important to note that Lemma 12 implies that using Algorithm 2 with LP and the threshold function f_α cannot achieve an approximation better than $11/6$ with respect to the makespan. To this end we strengthen LP using the following constraint (which we denote by Set constraints):

$$\sum_{e \in S} x_{e,u} \leq |S| - 1 \quad \forall u \in V, \forall S \subseteq \delta(u) : S \notin \mathcal{F}(u) \text{ and } |S| \leq k \quad (\text{Set})$$

We call the new relaxation LP_k .⁸ Intuitively, the Set constraints enforce that given an infeasible set of edges S touching u not all edges of S can be oriented toward u . In fact, for our specific choice of a threshold function f we use $k = 3$. Thus, no separation oracle is required when solving the relaxation. The exact result is formulated in the following theorem:

► **Theorem 13.** *There exists a rounding function f such that Algorithm 2 finds an orientation that is a $(1.75 + \gamma, 1/(2\gamma + 0.5))$ -approximation, for every $1/12 - \epsilon/2 \leq \gamma \leq 1/12$, where $\epsilon = \sqrt{33}/2 - 17/6$.*

Note that this theorem extends the tradeoff achieved in Theorem 10, and together both theorems achieve the tradeoff of Theorem 1. The threshold function f_ϵ we use in the proof of Theorem 13 is defined as follows:

$$f_\epsilon(p_e) = \begin{cases} 2/3 - \epsilon & \text{if } p_e > 1/2 \\ 2/3 + \epsilon/2 & \text{if } 1/3 < p_e \leq 1/2 \\ 1 & \text{if } p_e \leq 1/3 \end{cases} \quad (2)$$

where $0 \leq \epsilon \leq \sqrt{33}/2 - 17/6$. The following lemma upper bounds the makespan:

► **Lemma 14.** *The output of Algorithm 2 with the threshold function f_ϵ , has a makespan of at most $11/6 - \epsilon/2$.*

Now we conclude with the proofs of Theorems 13 and 1.

Proof of Theorem 13. Follows immediately from Lemmas 14 and 9, and choosing $\gamma = 1/12 - \epsilon/2$. ◀

Proof of Theorem 1. Follows immediately from Theorems 10 and 13. ◀

⁸ Similarly to LP , for some of the extensions of GB we add that $x_{e,u} = 0$ if $p_{e,u} > 1$ (for every $e \in E$ and $u \in e$).

4 Lower Bound on The Tradeoff Between Makespan and Cost

We show that using LP_k for every $k \in \mathbb{N}$, one must loose in the total orientation cost when obtaining an approximation for the makespan that is strictly better than 2. This is in contrast to the classic result of [15] for which one can achieve an approximation factor of 2 with respect to the makespan with no loss in the assignment cost. This result is formulated in Theorem 3.

5 Extending Graph Balancing to Hyperedges and Unrelated Weights

5.1 Graph Balancing with Unrelated Light Hyperedges

Let us recall the definition of $\text{GBUH}(\beta)$, where $\beta \in [0, 1]$. The input consists of a hypergraph, where each vertex represents a machine and each hyperedge represents a job. The jobs are of two types, “light” and “heavy”. Every light hyperedge $e \in E$ is associated with weights $p_{e,u}$, one for each vertex $u \in e$ (i.e., e is unrelated since it has a different processing time for each of the machines it can be assigned to). The requirement is that $p_{e,u} \leq \beta$ for every $u \in e$. On the other hand, every heavy hyperedge $e \in E$ must in fact be an edge, i.e., $|e| = 2$. Such a heavy e is associated with a single weight $p_e \in [0, 1]$ (i.e., e is related since it has the same processing time for each of the two machines it can be assigned to). In the above, as previously mentioned, we assume without loss of generality that the largest weight equals 1. For both types, light and heavy, orienting e toward one of its endpoints is equivalent to assigning the job e represents to the machine that is represented by the vertex e was oriented to. It is important to note that when $\beta = 1$ the problem is exactly GAP, and when $\beta = 0$ the problem is exactly GB.

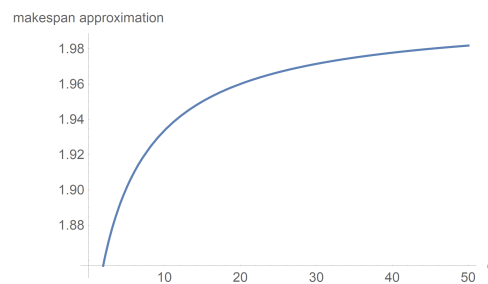
Our result for $\text{GBUH}(\beta)$ is summarized in Theorem 4, which improves upon the previous result of [6] (refer to Section 1 for a thorough discussion on how our result improves upon [6]). To the best of our knowledge, our result provides the first approximation better than 2 with respect to the makespan of a natural problem that captures GB but is not captured by RA.

5.2 Graph Balancing with Unrelated Light Hyperedges and Unrelated Heavy Edges

The problem of $\text{GBU}(\beta)$ further generalizes the above $\text{GBUH}(\beta)$ as it allows heavy edges to have unrelated weights. Formally, every heavy edge $e = (u, v) \in E$ is associated with two weights $p_{e,u}$ and $p_{e,v}$, i.e., e is unrelated since $p_{e,u}$ indicates the processing time of the job e represents on the machine that is represented by u . The requirement is that $p_{e,u}, p_{e,v} \in (\beta, 1]$. As mentioned earlier, it is assumed that the value of the optimal makespan is at least 1 (otherwise the problem is as hard as GAP). Our results relating to $\text{GBU}(\beta)$ are formulated in Theorems 5 and 6.

5.3 Semi-Related Graph Balancing

Consider the general problem of UNRELATED GRAPH BALANCING, which is identical to GB except that an edge can have a different weight depending on its orientation: $p_{e,u}$ and $p_{e,v}$ for every $e = (u, v) \in E$, i.e., the weights are unrelated. This generalization of GB was already considered in [17, 4], who presented lower bounds for the problem. Specifically, they showed that the even the configuration LP (which captures LP_k) has an integrality gap of 2 with respect to the makespan.



■ **Figure 4** Makespan approximation as a function of the value c .

We consider an interesting special case of the above problem where the weights are still unrelated, but cannot vary arbitrarily. Formally, each edge $e = (u, v) \in E$ has two weights depending on the vertex e is oriented to, which satisfy: $p_{e,u} \leq c \cdot p_{e,v}$ and $p_{e,v} \leq c \cdot p_{e,u}$ (where $c \geq 1$ is a parameter of the problem). We denote this problem by SEMI-RELATED GRAPH BALANCING (SRGB(c)).

Our result for SRGB(c) is formulated in Theorem 7. Note that SRGB(c) captures GB when $c = 1$, and indeed in Theorem 7 we achieve a $(11/6, 3/2)$ -approximation for SRGB(c) when $c = 1$ (similarly to Theorem 10). Moreover, when $c = \infty$ Theorem 7 achieves a $(2, 1)$ -approximation for SRGB(c), matching the integrality gap of [17, 4]. Finally, we also show that in general Theorem 7 provides a $(2 - \Omega(1/c), 1 + O(1/c))$ -approximation for SRGB(c).

Figure 4 shows the makespan approximation obtained in Theorem 7 as a function of c .

In order to prove Theorem 7 we use Algorithm 2 and LP_k (replacing p_e with $p_{e,u}$) with a suitable choice of a threshold function f .

References

- 1 Nikhil Bansal and Maxim Sviridenko. The Santa Claus problem. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 31–40, 2006. doi:10.1145/1132516.1132522.
- 2 Deeparnab Chakrabarty and Kirankumar Shiragur. Graph Balancing with Two Edge Types. *CoRR*, abs/1604.06918, 2016. arXiv:1604.06918.
- 3 Tomas Ebenlendr, Marek Krcal, and Jiri Sgall. Graph balancing: a special case of scheduling unrelated parallel machines. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 483–490, 2008. URL: <http://dl.acm.org/citation.cfm?id=1347082.1347135>.
- 4 Tomas Ebenlendr, Marek Krcal, and Jiri Sgall. Graph Balancing: A Special Case of Scheduling Unrelated Parallel Machines. *Algorithmica*, 68(1):62–80, 2014. doi:10.1007/s00453-012-9668-9.
- 5 Ellis Horowitz and Sartaj Sahni. Exact and Approximate Algorithms for Scheduling Nonidentical Processors. *J. ACM*, 23:317–327, April 1976.
- 6 Chien-Chung Huang and Sebastian Ott. A Combinatorial Approximation Algorithm for Graph Balancing with Light Hyper Edges. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 49:1–49:15, 2016. doi:10.4230/LIPIcs.ESA.2016.49.
- 7 Klaus Jansen, Kati Land, and Marten Maack. Estimating the Makespan of the Two-Valued Restricted Assignment Problem. *Algorithmica*, 80(4):1357–1382, 2018. doi:10.1007/s00453-017-0314-4.
- 8 Klaus Jansen and Lorant Porkolab. Improved Approximation Schemes for Scheduling Unrelated Parallel Machines. *Mathematics of Operations Research*, 26(2):324–338, 2001.

- 9 Klaus Jansen and Lars Rohwedder. On the Configuration-LP of the Restricted Assignment Problem. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2670–2678, 2017. doi:10.1137/1.9781611974782.176.
- 10 Klaus Jansen and Lars Rohwedder. Local search breaks 1.75 for Graph Balancing. *CoRR*, abs/1811.00955, 2018. arXiv:1811.00955.
- 11 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.
- 12 Daniel R. Page and Roberto Solis-Oba. A $3/2$ -Approximation Algorithm for the Graph Balancing Problem with Two Weights. *Algorithms*, 9(2):38, 2016. doi:10.3390/a9020038.
- 13 Dorit S. Hochbaum and David Shmoys. A Polynomial Approximation Scheme for Scheduling on Uniform Processors: Using the Dual Approximation Approach. *SIAM J. Comput.*, 17:539–551, June 1988.
- 14 Evgeny V. Shchepin and Nodari Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 33(2):127–133, 2005.
- 15 David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62:461–474, 1993. doi:10.1007/BF01585178.
- 16 Ola Svensson. Santa Claus schedules jobs on unrelated machines. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 617–626, 2011. doi:10.1145/1993636.1993718.
- 17 José Verschae and Andreas Wiese. On the configuration-LP for scheduling on unrelated machines. *J. Scheduling*, 17(4):371–383, 2014. doi:10.1007/s10951-013-0359-4.
- 18 Chao Wang and René Sitters. On some special cases of the restricted assignment problem. *Inf. Process. Lett.*, 116(11):723–728, 2016. doi:10.1016/j.ipl.2016.06.007.
- 19 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. URL: http://www.cambridge.org/de/knowledge/isbn/item5759340/?site_locale=de_DE.

FPT-Algorithms for Computing Gromov-Hausdorff and Interleaving Distances Between Trees

Elena Farahbakhsh Touli¹

Stockholm University, Sweden

<https://www.su.se/english/profiles/elfa1534-1.428605>

elena.touli@math.su.se

Yusu Wang¹

The Ohio State University, USA

<https://web.cse.ohio-state.edu/~wang.1016/>

yusu@cse.ohio-state.edu

Abstract

The Gromov-Hausdorff distance is a natural way to measure the distortion between two metric spaces. However, there has been only limited algorithmic development to compute or approximate this distance. We focus on computing the Gromov-Hausdorff distance between two metric trees. Roughly speaking, a metric tree is a metric space that can be realized by the shortest path metric on a tree. Any finite tree with positive edge weight can be viewed as a metric tree where the weight is treated as edge length and the metric is the induced shortest path metric in the tree. Previously, Agarwal et al. showed that even for trees with unit edge length, it is NP-hard to approximate the Gromov-Hausdorff distance between them within a factor of 3. In this paper, we present a fixed-parameter tractable (FPT) algorithm that can approximate the Gromov-Hausdorff distance between two general metric trees within a *multiplicative factor* of 14.

Interestingly, the development of our algorithm is made possible by a connection between the Gromov-Hausdorff distance for metric trees and the interleaving distance for the so-called merge trees. The merge trees arise in practice naturally as a simple yet meaningful topological summary (it is a variant of the Reeb graphs and contour trees), and are of independent interest. It turns out that an exact or approximation algorithm for the interleaving distance leads to an approximation algorithm for the Gromov-Hausdorff distance. One of the key contributions of our work is that we re-define the interleaving distance in a way that makes it easier to develop dynamic programming approaches to compute it. We then present a fixed-parameter tractable algorithm to compute the interleaving distance between two merge trees **exactly**, which ultimately leads to an FPT-algorithm to approximate the Gromov-Hausdorff distance between two metric trees. This exact FPT-algorithm to compute the interleaving distance between merge trees is of interest itself, as it is known that it is NP-hard to approximate it within a factor of 3, and previously the best known algorithm has an approximation factor of $O(\sqrt{n})$ even for trees with unit edge length.

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

Keywords and phrases Gromov-Hausdorff distance, Interleaving distance, Merge trees

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.83

Funding This work is partially supported by National Science Foundation (NSF) under grants CCF-1740761, IIS-1815697 and CCF-1618247, as well as by National Institute of Health (NIH) under grant R01EB022899.

Acknowledgements We thank reviewers for helpful comments. We would like to thank Kyle Fox, for suggesting an elegant double-binary search procedure, to improve the time complexity of the optimal interleaving distance by almost a factor of n^2 (see Theorem 19), which further leads to a similar improvement for approximating the Gromov-Hausdorff distance (Theorem 20).

¹ Corresponding author



1 Introduction

Given two metric spaces (X, d_X) and (Y, d_Y) , a natural way to measure their distance is via the *Gromov-Hausdorff distance* $\delta_{\mathcal{GH}}(X, Y)$ between them [15], which intuitively describes how much *additive* distance distortion is needed to make the two metric spaces isometric.

We are interested in computing the Gromov-Hausdorff distance between *metric trees*. Roughly speaking, a metric tree (X, d) is a geodesic-metric space that can be realized by the shortest path metric on a tree. Any finite tree $T = (V, E)$ with positive edge weights $w : E \rightarrow \mathbb{R}$ can be naturally viewed as a metric tree $\mathcal{T} = (|T|, d)$: the space is the underlying space $|T|$ of T , each edge e can be viewed as a segment of length $w(e)$, and the distance d is the induced shortest path metric. See Figure 1 (a) for an example. Metric trees occur commonly in practical applications: e.g., a neuron cell has a tree morphology, and can be modeled as an embedded metric tree in \mathbb{R}^3 . It also represents an important family of metric spaces that has for example attracted much attention in the literature of metric embedding and recovery of hierarchical structures, e.g., [2, 4, 3, 10, 11, 13, 23].

Unfortunately, it is shown in [1, 22] that it is not only NP-hard to compute the Gromov-Hausdorff distance between two trees, but also NP-hard to approximate it within a factor of 3 even for trees with unit edge length. A polynomial-time approximation algorithm is given in [1]; however, the approximation factor is high: it is $O(\sqrt{n})$ even for unit-edge weight trees.

Another family of tree structures that is of practical interest is the so-called *merge tree*. Intuitively, a merge tree is a rooted tree T associated with a real-valued function $f : T \rightarrow \mathbb{R}$ such that the function value is monotonically decreasing along any root-to-leaf path – We can think of a merge tree to be a tree with height function associated to it where all nodes with degree > 2 are down-forks (merging nodes); see Figure 1 (b). The merge tree is a loop-free variant of the so-called Reeb graph, which is a simple yet meaningful topological summary for a scalar field $g : X \rightarrow \mathbb{R}$ defined on a domain X , and has been widely used in many applications in graphics and visualization e.g., [7, 14, 17, 24]. Morozov et al. introduced the *interleaving distance* to compare merge trees [20], based on a natural “interleaving idea” which has recently become fundamental in comparing various topological objects. Also, several distance measures have been proposed for the Reeb graphs [5, 6, 12]. When applying them to merge trees, it turns out that two of these distance measures are equivalent to the interleaving distance. However, the same reduction in [1] to show the hardness of approximating the Gromov-Hausdorff distance can also be used to show that it is NP-hard to approximate the interleaving distance between two merge trees within a factor 3.

New work

Although the Gromov-Hausdorff distance is a natural way to measure the degree of near-isometry between metric spaces [15, 19], the algorithmic development for it has been very limited so far [1, 9, 21, 22]. In [22], Schmiedl gave an FPT algorithm for approximating the Gromov-Hausdorff distance between two *finite metrics*, where the approximation contains *both an additive and multiplicative terms*; see more discussion in *Remarks* after Theorem 20. In this paper, we present the first FPT algorithm to approximate the Gromov-Hausdorff distance for metric trees *within a constant multiplicative factor*.

Interestingly, the development of our approximation algorithm is made possible via a connection between the Gromov-Hausdorff distance between metric trees and the interleaving distance between certain merge trees (which has already been observed previously in [1]). This connection implies that any exact or approximation algorithm for the interleaving distance will lead to an approximation algorithm for the Gromov-Hausdorff distance for

metric trees of similar time complexity. Hence we can focus on developing algorithms for the interleaving distance. The original interleaving distance definition requires considering a pair of maps between the two input merge trees and their interaction. One of the key insights of our work is that we can in fact develop an equivalent definition for the interleaving distance that relies on only *a single map* from one tree to the other. This, together with the height functions equipped with merge trees (which give rises to natural ordering between points in the two trees), essentially allows us to develop a dynamic programming algorithm to check whether the interleaving distance between two merge trees is smaller than a given threshold or not: In particular in Section 4, we first give a simpler DP algorithm with slower time complexity to illustrate the main idea. We then show how we can modify this DP algorithm to improve the time complexity. Finally, we solve the optimization problem for computing the interleaving distance² in Section 5, which leads to a constant-factor (a multiplicative factor of 14) approximation FPT algorithm for the Gromov-Hausdorff distance between metric trees.

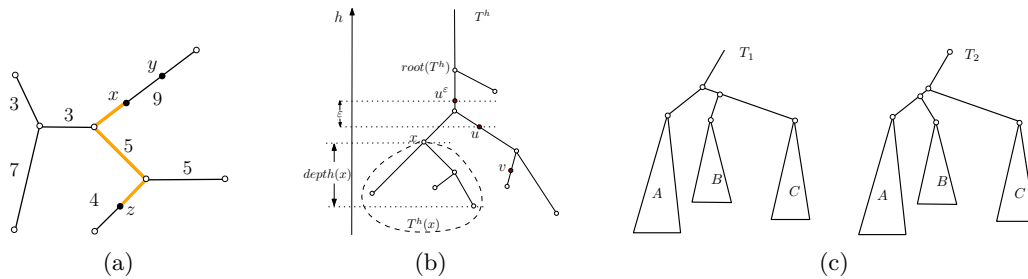


Figure 1 (a) A metric tree (T, d_T) with edge length marked. Tree nodes are white dots. $d_T(x, z) = 3 + 5 + 2 = 10$ is the length of the thickened path $\pi(x, z)$. (b) A merge tree T^h , with examples of $u \succ v$, u^ε , $T^h(x)$ and $depth(x)$ marked. (c) Tree alignment distance between T_1 and T_2 arbitrarily large, while $\delta_{\mathcal{GH}}(T_1, T_2)$ is roughly bounded by the pairwise distance difference which is small.

More on related work

There have been several tree distances proposed in the literature. Two most well-known ones are the tree edit and tree alignment distances [8], primarily developed to compare labeled trees. Unfortunately, both distances are MAX SNP-hard to compute for un-ordered trees [18, 26]. For tree edit distance, it is MAX SNP-hard even for trees with bounded degree. For the tree alignment distance, it can be computed in polynomial time for trees with bounded degree. However the tree alignment distance requires that parent-child relation to be strictly preserved, and thus the small local configuration change shown in Figure 1 (c) will incur a large tree alignment distance.

We will not survey the large literature in metric embedding which typically minimizes the metric distortion in a *multiplicative* manner. However, we mention the work of Hall and Papadimitriou [16], where, given two equal-sized point sets, they propose to find the best *bijection* under which the *additive distortion* is minimized. They show it is NP-hard to approximate this optimal additive distortion within a factor of 3 even for points in \mathbb{R}^3 .

² We note that the final time complexity for the optimization problem presented in Theorem 19 is based on an argument by Kyle Fox. His argument improves our previous n^4 factor (as in Theorem 18) by an almost n^2 factor, by performing a double-binary search, instead of a sequence search we originally used.

In contrast, the Gromov-Hausdorff distance is also *additive*, but allows for many-to-many correspondence (instead of bijection) between points from two input metric spaces. We also note that our metric trees consist of all points in the underlying space of input trees (i.e., including points in the interior of a tree edge). This makes the distance robust against adding extra nodes and short “hairs” (branches). Nevertheless, we can also consider discrete metric trees, where we only aim to align nodes of input trees (instead of all points in the underlying space of the trees). Our algorithms hold for the discrete case as well.

2 Preliminaries

Metric space, metric trees

A metric space is a pair (X, d) where X is a set and $d : X \times X \rightarrow \mathbb{R}_{\geq 0}$ satisfies: (i) for any $x, y \in X$, $d(x, y) \geq 0$ and $d(x, y) = 0$ holds only when $x = y$; (ii) $d(x, y) = d(y, x)$, and (iii) for any x, y, z , $d(x, z) \leq d(x, y) + d(y, z)$. We call d a metric on the space X . A metric space (X, d) is a finite metric tree if it is a length metric space³ and X is homeomorphic to the underlying space $|T|$ of some finite tree $T = (V, E)$.

Equivalently, suppose we are given a finite tree $T = (V, E)$ where each edge $e \in E$ has a positive weight $\ell(e) > 0$. View the underlying space $|e|$ of e as a segment with length $\ell(e)$ (i.e., it is isometric to $[0, \ell(e)]$), and we can thus define the distance $d_T(x, y)$ between any two points $x, y \in |e|$ as the length of the sub-segment $e[x, y]$. The underlying space $|T|$ of T is the union of all these segments (and thus includes points in the interior of each edge as well). For any $x, z \in |T|$, there is a unique simple path $\pi(x, z) \subset |T|$ connecting them. The (shortest path) distance $d_T(x, z)$ equal to the length of this path, which is simply the sum of the lengths of the restrictions of this path to edges in T . See Figure 1 (a). The space $|T|$ equipped with d_T is a metric tree $(|T|, d_T)$.

Given a tree $T = (V, E)$, we use the term *tree nodes* to refer to points in V , and an arbitrary $x \in |T|$ potentially from the interior of some tree edge is referred to as a *point*. Given T , we also use $V(T)$ and $E(T)$ to denote its node-set and edge-set, respectively. To emphasize the combinatorial structure behind a metric tree, in the paper we will write a metric tree (T, d_T) , with the understanding that the space is in fact the underlying space $|T|$ of T .

Note that if we restrict this metric space to only the tree nodes, we obtain a *discrete metric tree* $(V(T), d_T)$, and the distance between two tree nodes is simply the standard shortest path distance between them in a weighted graph (tree T in this case). Our algorithms developed in this paper can be made to work for the discrete metric trees as well.

Gromov-Hausdorff distance

Given two metric spaces $\mathcal{X} = (X, d_X)$ and $\mathcal{Y} = (Y, d_Y)$, a *correspondence* between them is a relation $C : X \times Y$ whose projection on X and on Y are both surjective; i.e., for any $x \in X$, there is at least one $(x, y) \in C$, and for any $y' \in Y$, there is at least one $(x', y') \in C$. If $(x, y) \in C$, then we say y (resp. x) is a pairing partner for x (resp. y); note that x (resp. y) could have multiple pairing partners. The cost of this correspondence is defined as:

$$\text{cost}(C) = \max_{(x, y), (x', y') \in C} |d_X(x, x') - d_Y(y, y')|,$$

which measures the maximum metric distortion (difference in pairwise distances) under this correspondence. The *Gromov-Hausdorff distance* between them is:

$$\delta_{\text{GH}}(\mathcal{X}, \mathcal{Y}) = \frac{1}{2} \inf_{C \in \Pi(\mathcal{X}, \mathcal{Y})} \text{cost}(C), \quad \text{where } \Pi(\mathcal{X}, \mathcal{Y}) = \text{set of correspondences between } X \text{ and } Y.$$

³ (X, d) is a length metric space if d is the same as the shortest path (i.e., intrinsic) metric it induces on X .

Merge trees

A *merge tree* is a pair (T, h) where T is a rooted tree, and the continuous function $h : |T| \rightarrow \mathbb{R}$ is *monotone* in the sense the value of h is decreasing along any root-to-leaf path. See Figure 1 (b) for an example. For simplicity, we often write the merge tree as T^h , and refer to h as the *height function*, and $h(x)$ the *height* of a point $x \in |T|$. The merge tree is a natural object: e.g., it can be used to model a hierarchical clustering tree, where the height of a tree node indicates the parameter when the cluster (corresponding to the subtree rooted at this node) is formed. It also arises as a simple topological summary of a scalar function $\tilde{h} : M \rightarrow \mathbb{R}$ on a domain M , which tracks the connected component information of the sub-level sets $\tilde{h}^{-1}(-\infty, a]$ as $a \in \mathbb{R}$ increases.

To define the interleaving distance, we modify a merge tree T^h slightly by extending a ray from $\text{root}(T^h)$ upwards with function value h goes to $+\infty$. All merge trees from now on refer to this modified version. Given a merge tree T^h and a point $x \in |T|$, $T^h(x)$ is the subtree of T^h rooted at x , and the *depth of x (or of $T^h(x)$)*, denoted by $\text{depth}(x)$, is the largest function value difference between x and any node in its subtree; that is, the height of the entire subtree $T^h(x)$ w.r.t. function h . Given any two points $u, v \in |T|$, we use $u \succeq v$ to denote that u is an ancestor of v ; $u \succ v$ if u is an ancestor of v and $u \neq v$. Similarly, $v \preceq u$ means that v is a descendant of u . Also, the degree of a node in a merge tree is defined as the downward degree of the node. We use $LCA(u, v)$ to represent the lowest common ancestor of u and v in $|T|$. For any non-negative value $\varepsilon \geq 0$, u^ε represents the unique ancestor of u in T such that $h(u^\varepsilon) - h(u) = \varepsilon$. See Figure 1 (b).

Interleaving distance

We now define the interleaving distance between two merge trees T_1^f and T_2^g , associated with functions $f : |T_1^f| \rightarrow \mathbb{R}$ and $g : |T_2^g| \rightarrow \mathbb{R}$, respectively.

► **Definition 1** (ε -Compatible maps [20]). *A pair of continuous maps $\alpha : |T_1^f| \rightarrow |T_2^g|$ and $\beta : |T_2^g| \rightarrow |T_1^f|$ is ε -compatible w.r.t T_1^f and T_2^g if the following four conditions hold:*

- (C1). $g(\alpha(u)) = f(u) + \varepsilon$ and (C2). $\beta \circ \alpha(u) = u^{2\varepsilon}$ for any $u \in |T_1^f|$;
 (C3). $f(\beta(w)) = g(w) + \varepsilon$ and (C4). $\alpha \circ \beta(w) = w^{2\varepsilon}$ for any $w \in |T_2^g|$.

To provide some intuition for this definition, note that if $\varepsilon = 0$, then $\alpha = \beta^{-1}$: In this case, the two trees T_1 and T_2 are not only isomorphic, but also the function values associated to them are preserved under the isomorphism. In general for $\varepsilon > 0$, this quantity measures how far a pair of maps are away from forming a function-preserving isomorphism between T_1^f and T_2^g . In particular, β is no longer the inverse of α . However, the two maps relate to each other in the sense that if we send a point $u \in |T_1^f|$ to $|T_2^g|$ through $\alpha : |T_1^f| \rightarrow |T_2^g|$, then bring it back via $\beta : |T_2^g| \rightarrow |T_1^f|$, we come back at an ancestor of u in $|T_1^f|$ (i.e, property (C2)). This ancestor must be at height $f(u) + 2\varepsilon$ due to properties (C1) and (C3).

► **Definition 2** (Interleaving distance [20]). *The interleaving distance between two merge trees T_1^f and T_2^g is defined as:*

$$d_I(T_1^f, T_2^g) = \inf \{ \varepsilon \mid \text{there exist a pair of } \varepsilon\text{-compatible maps w.r.t } T_1^f \text{ and } T_2^g \}. \quad (1)$$

Interestingly, it is shown in [1] that the Gromov-Hausdorff distance between two metric trees is related to the interleaving distance between two specific merge trees.

▷ Claim 3 ([1]). Given two metric trees $\mathcal{T}_1 = (T_1, d_1)$ and $\mathcal{T}_2 = (T_2, d_2)$ with node sets $V_1 = V(T_1)$ and $V_2 = V(T_2)$, respectively, let $f_u : |T_1| \rightarrow \mathbb{R}$ (resp. $g_w : |T_2| \rightarrow \mathbb{R}$) denote the geodesic distance function to the base point $u \in V_1$ (resp. $v \in V_2$) defined by $f_u(x) = -d_1(x, u)$ for any $x \in |T_1|$ (resp. $g_w(y) = -d_2(y, w)$ for any $y \in |T_2|$). Set $\mu := \min_{u \in V_1, w \in V_2} d_I(T_1^{f_u}, T_2^{g_w})$. We then have that

$$\frac{\mu}{14} \leq \delta_{\mathcal{GH}}(\mathcal{T}_1, \mathcal{T}_2) \leq 2\mu.$$

Note that to compute the quantity μ , we only need to check all pairs of *tree nodes* of T_1 and T_2 , instead of all pairs of points from $|T_1|$ and $|T_2|$.

We say a quantity A is a c -*approximation* for a quantity B if $\frac{A}{c} \leq B \leq cA$; obviously, $c \geq 1$ and $c = 1$ means that $A = B$. The above claim immediately suggests the following:

► **Corollary 4.** *If there is an algorithm to c -approximate the interleaving distance between any two merge trees in $T(n)$ time, where n is the total complexity of input trees, then there is an algorithm to $O(c)$ -approximate the Gromov-Hausdorff distance between two metric trees in $n^2T(n)$ time.*

In the remainder of this paper, we will focus on developing an algorithm to compute the interleaving distance between two merge trees T_1^f and T_2^g . In particular, in Section 3 we will first show an equivalent definition for interleaving distance, which has a nice structure that helps us to develop a fixed-parameter tractable algorithm for the decision problem of “Is $d_I(T_1^f, T_2^g) \geq \varepsilon$?” in Section 4. We show how this ultimately leads to FPT algorithms to compute the interleaving distance **exactly** and to **approximate** the Gromov-Hausdorff distance in Section 5.

3 A New Definition for Interleaving Distance

Given two merge trees T_1^f and T_2^g and $\delta > 0$, to answer the question “Is $d_I(T_1^f, T_2^g) \leq \delta$?”, a natural idea is to scan the two trees bottom up w.r.t the “height” values (i.e. f and g), while checking for possible ε -compatible maps between the partial forests of T_1^f and T_2^g already scanned. However, the interaction between the pair maps α and β makes it complicated to maintain potential maps. We now show that in fact, we only need to check for the existence of a *single* map from T_1^f to T_2^g , which we will call the ε -good map. We believe that this result is of independent interest.

► **Definition 5** (ε -good map). *A continuous map $\alpha : |T_1^f| \rightarrow |T_2^g|$ is ε -good if and only if:*

- (P1) *for any $u \in |T_1^f|$, we have $g(\alpha(u)) = f(u) + \varepsilon$;*
- (P2) *if $\alpha(u_1) \succeq \alpha(u_2)$, then we have $u_1^{2\varepsilon} \succeq u_2^{2\varepsilon}$, (note $u_1 \succeq u_2$ may not be true); and*
- (P3) *if $w \in |T_2^g| \setminus \text{Im}(\alpha)$, then we have $|g(w^F) - g(w)| \leq 2\varepsilon$, where $\text{Im}(\alpha) \subseteq |T_2^g|$ is the image of α , and w^F is the lowest ancestor of w in $\text{Im}(\alpha)$.*

A map $\rho : |T_1^{h_1}| \rightarrow |T_2^{h_2}|$ between two arbitrary merge trees $T_1^{h_1}$ and $T_2^{h_2}$ is *monotone* if for any $u \in |T_1^{h_1}|$, we have that $h_2(\rho(u)) \geq h_1(u)$. In other word, ρ carries any point u from $T_1^{h_1}$ to a point higher than it in $T_2^{h_2}$. If ρ is continuous, then it will map an ancestor of u in $T_1^{h_1}$ to an ancestor of $\rho(u)$ in $T_2^{h_2}$ as stated below (but the converse is not necessarily true):

► **Observation 6.** *Given a continuous and monotone map $\rho : |T_1^{h_1}| \rightarrow |T_2^{h_2}|$ between two merge trees $T_1^{h_1}$ and $T_2^{h_2}$, we have that if $u_1 \succeq u_2$ in $T_1^{h_1}$, then $\rho(u_1) \succeq \rho(u_2)$ in $T_2^{h_2}$.*

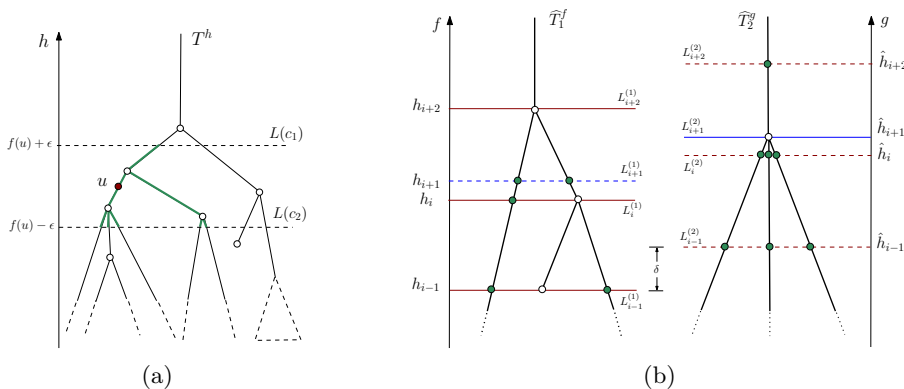
This implies that if $w = \rho(u)$ for $u \in |T_1^{h_1}|$, then ρ maps the subtree $T_1^{h_1}(u)$ rooted at u into the subtree $T_2^{h_2}(w)$ rooted at w . This also implies that if $w \notin \text{Im}(\rho)$, neither does any of its descendant.

Note that an ε -good map, or a pair of ε -compatible maps, are all monotone and continuous. Hence the above observations are applicable to all these maps.

The main result of this section is as follows. Its proof is in [25].

► **Theorem 7.** *Given any two merge trees T_1^f and T_2^g , then $d_I(T_1^f, T_2^g) \leq \varepsilon$ if and only if there exists an ε -good map $\alpha : |T_1^f| \rightarrow |T_2^g|$.*

4 Decision Problem for Interleaving Distance



■ **Figure 2** (a) Green component within the slab is $B_\varepsilon(u, T_1^f)$. The sum of degrees for nodes within this ε -ball is 13. The ε -degree bound $\tau_\varepsilon(T_1^f, T_2^g)$ is the largest value of this sum for any ε -ball in T_1^f or in T_2^g . (b) White dots are tree nodes of T_1^f and T_2^g . Green dots are newly augmented tree nodes in \widehat{T}_1^f and \widehat{T}_2^g .

In this section, given two merge trees T_1^f and T_2^g as well as a positive value $\delta > 0$, we aim to develop a fixed-parameter tractable algorithm for the decision problem “Is $d_I(T_1^f, T_2^g) \leq \delta$?”. The specific parameter our algorithm uses is the following: Given a merge tree T^h and any point $u \in T^h$, let $B_\varepsilon(u; T^h)$ denote the ε -ball

$$B_\varepsilon(u; T^h) = \{x \in |T| \mid \forall y \in \pi_T(u, x), |h(y) - h(u)| \leq \varepsilon\},$$

where $\pi_T(u, x)$ is the unique path from u to x in T^h . In other words, $B_\varepsilon(u; T^h)$ contains all points reachable from u via a path whose function value is completely contained with the range $[f(u) - \varepsilon, f(u) + \varepsilon]$. See Figure 2 (a) for an example: in particular, consider the restriction of T^h within the height interval $[f(u) - \varepsilon, f(u) + \varepsilon]$. There could be multiple components within this slab, and $B_\varepsilon(u; T^h)$ is the component containing u .

Parameter τ_δ : Let $\tau_\varepsilon(T_1^f, T_2^g)$ denote the largest sum of degrees of all tree nodes contained in any ε -ball in T_1^f or T_2^g , which we also refer to as the ε -degree-bound of T_1^f and T_2^g . The parameter for our algorithm for the decision problem will be $\tau_\delta = \tau_\delta(T_1^f, T_2^g)$.

4.1 A Slower FPT-Algorithm

Augmented trees

We now develop an algorithm for the decision problem “Is $d_I(T_1^f, T_2^g) \leq \delta$?” via a dynamic programming type approach. First, we will show that, even though a δ -good map is defined for all (infinite number of) points from T_1^f and T_2^g , we can check for its existence by inspecting

only a finite number of points from T_1^f and T_2^g . In particular, we will augment the input merge trees T_1^f and T_2^g with extra tree nodes, and our algorithm later only needs to consider the discrete nodes in these augmented trees to answer the decision problem.

The set of points from tree T_1^f or T_2^g at a certain height value c is called a *level* (at height c), denoted by $L(c)$. For example, in Figure 2 (a), the level $L(c_1)$ for $c_1 = f(u) + \varepsilon$, contains 2 points, while $L(c_2)$ with $c_2 = f(u) - \varepsilon$ contains 7 points. The function value of a level L is called its *height*, denoted by $height(L)$; so $height(L(c)) = c$.

► **Definition 8** (Critical-heights and Super-levels). *For the tree T_1^f , the set of critical-heights C_1 consists of the function values of all tree nodes of T_1^f ; similarly, define C_2 for T_2^g . That is,*

$$C_1 := \{f(x) \mid x \text{ is a tree node of } T_1^f\}; \text{ and } C_2 := \{g(y) \mid y \text{ is a tree node of } T_2^g\}.$$

The set of super-levels \mathcal{L}_1 w.r.t. δ for T_1^f and the set of super-levels \mathcal{L}_2 for T_2^g are:

$$\begin{aligned} \mathcal{L}_1 &:= \{L(c) \mid c \in C_1\} \cup \{L(c - \delta) \mid c \in C_2\} \text{ while} \\ \mathcal{L}_2 &:= \{L(c + \delta) \mid c \in C_1\} \cup \{L(c) \mid c \in C_2\}. \end{aligned}$$

Now sort all levels in \mathcal{L}_i in increasing order of their heights, denoted by $\mathcal{L}_1 = \{L_1^{(1)}, L_2^{(1)}, \dots, L_m^{(1)}\}$ and $\mathcal{L}_2 = \{L_1^{(2)}, \dots, L_m^{(2)}\}$, respectively. The *child-level* of super-level $L_i^{(1)}$ (resp. $L_i^{(2)}$) is $L_{i-1}^{(1)}$ (resp. $L_{i-1}^{(2)}$) for any $i \in [2, m]$; symmetrically, $L_i^{(1)}$ (resp. $L_i^{(2)}$) is the *parent-level* of $L_{i-1}^{(1)}$ (resp. $L_{i-1}^{(2)}$). Let h_1, \dots, h_m be the sequence of height values for $L_1^{(1)}, L_2^{(1)}, \dots, L_m^{(1)}$; that is, $h_i = height(L_i^{(1)})$. Similarly, let $\hat{h}_1, \hat{h}_2, \dots, \hat{h}_m$ be the corresponding sequence for $L_i^{(2)}$'s.

Note that there is a one-to-one correspondence between super-levels in \mathcal{L}_1 and \mathcal{L}_2 : specifically, for any $i \in [1, m]$, we have $\hat{h}_i = h_i + \delta$. From now on, when we refer to the i -th super-levels of \hat{T}_1^f and \hat{T}_2^g , we mean super-levels $L_i^{(1)}$ and $L_i^{(2)}$. Also observe that there is no tree node in between any two consecutive super-levels in either T_1^f or in T_2^g (all tree nodes are from some super-levels). See Figure 2 (b) for an illustration.

Next, we augment the tree T_1^f (resp. T_2^g) to add points from all super-levels from \mathcal{L}_1 (resp. from \mathcal{L}_2) also as *tree nodes*. The resulting *augmented trees* are denoted by \hat{T}_1^f and \hat{T}_2^g respectively; obviously, \hat{T}_1^f (resp. \hat{T}_2^g) has isomorphic underlying space as T_1^f (resp. T_2^g), just with additional degree-2 tree nodes. In particular, $V(\hat{T}_1^f)$ (resp. $V(\hat{T}_2^g)$) is formed by all points from all super-levels in \mathcal{L}_1 (resp. \mathcal{L}_2). See Figure 2 (b): In this figure, solid horizontal lines indicate levels passing through critical heights, while dashed ones are induced by critical height from the other tree. In what follows, given a super-level L , we use $V(L)$ to denote the set of nodes from this level. Note that $V(L_m^{(1)})$ and $V(L_m^{(2)})$ each contain only one node, which is $root(\hat{T}_1^f)$ and $root(\hat{T}_2^g)$ respectively. Given a node v from $L_i^{(1)}$ (resp. $L_i^{(2)}$), let $Ch(v)$ denote its children nodes in the augmented tree. Each child node of v must be from level $L_{i-1}^{(1)}$ (resp. $L_{i-1}^{(2)}$), as there are no tree-nodes between two consecutive super-levels.

► **Definition 9** (Valid pair). *Given a node $w \in V(\hat{T}_2^g)$ and a collection of nodes $S \subseteq V(\hat{T}_1^f)$, we say that (S, w) form a valid pair if there exists an index $j \in [1, m]$ such that (1) $S \subseteq V(L_j^{(1)})$ and $w \in V(L_j^{(2)})$ (which implies that nodes in S at height h_j while w has height $g(w) = \hat{h}_j$); and (2) all nodes in S have the same ancestor at height $h_j + 2\delta$ (which also equals $\hat{h}_j + \delta$). Intuitively, it indicates that S has the basic condition to be mapped to w under some ε -good maps.*

We say that S is valid if it participates some valid pair (and thus condition (2) above holds).

A first (slower) dynamic programming algorithm

We now describe our dynamic programming algorithm. To illustrate the main idea, we first describe a much cleaner but also slower dynamic programming algorithm $\text{DPgoodmap}()$ below. Later in Section 4.2 we modify this algorithm to improve its time complexity (which requires significant additional technical details).

Our algorithm maintains a certain quantity, called *feasibility* $F(S, w)$ for valid pairs in a bottom-up manner. Recall that we have defined the depth of a node $u \in T^h$ in a merge tree T^h as the height of the subtree $T^h(u)$ rooted at u ; or equivalently $\text{depth}(u) = \max_{x \preceq u} |h(u) - h(x)|$.

■ **Algorithm 1** $\text{DPgoodmap}(T_1^f, T_2^g, \delta)$.

Base case ($i = 1$): For each valid-pair (S, w) from level-1, set $F(S, w) = 1$ (“true”) if and only if $\text{depth}(w) \leq 2\delta$; otherwise, set $F(S, w) = 0$ (“false”).

When $i > 1$: Suppose we have already computed the feasibility values for all valid-pairs from level- $(i-1)$ or lower. Now for any valid-pair (S, w) from level- i , we set $F(S, w) = 1$ if and only if the following holds: Consider the set of children $\text{Ch}(S) \subseteq L_{i-1}^{(1)}$ of nodes in S , and w ’s children $\text{Ch}(w) = \{w_1, \dots, w_k\}$ in $L_{i-1}^{(2)}$.

If $\text{Ch}(w)$ is empty, then $F(S, w) = 1$ *only if* $\text{Ch}(S)$ is also empty; otherwise $F(S, w) = 0$.

If $\text{Ch}(w)$ is not empty, then we set $F(S, w) = 1$ if there exists a partition of $\text{Ch}(S) = S_1 \cup S_2 \cup \dots \cup S_k$ (where $S_i \cap S_j = \emptyset$ for $i \neq j$, and it is possible that $S_i = \emptyset$) such that for each $j \in [1, k]$,

(F-1) if $S_j \neq \emptyset$, then $F(S_j, w_j) = 1$; and

(F-2) if $S_j = \emptyset$, then $\text{depth}(w_j) \leq 2\delta - (\hat{h}_i - \hat{h}_{i-1})$; note that this implies that $\hat{h}_i - \hat{h}_{i-1} \leq 2\delta$ in this case.

Output: $\text{DPgoodmap}(T_1^f, T_2^g, \delta)$ returns “yes” if and only if $F(\text{root}(\hat{T}_1^f), \text{root}(\hat{T}_2^g)) = 1$.

Recall that $\text{root}(\hat{T}_1^f)$ (resp. $\text{root}(\hat{T}_2^g)$) is the only node in $V(L_m^{(1)})$ (resp. $V(L_m^{(2)})$).

We will first prove the following theorem for this slower. In Section 4.2 we show that time complexity can be reduced by almost a factor of n .

► Theorem 10.

- (i) Algorithm $\text{DPgoodmap}(T_1^f, T_2^g, \delta)$ returns “yes” if and only if $d_I(T_1^f, T_2^g) \leq \delta$.
- (ii) Algorithm $\text{DPgoodmap}(T_1^f, T_2^g, \delta)$ can be implemented to run in $O(n^3 2^\tau \tau^{\tau+1})$ time, where n is the total size of T_1^f, T_2^g , and $\tau = \tau_\delta(T_1^f, T_2^g)$ is the δ -degree-bound w.r.t. T_1^f and T_2^g .

Note that if τ is constant, then the time complexity is $O(n^3)$.

In the remainder of this section, we sketch the proof of Theorem 10.

Part (i) of Theorem 10: correctness

We first show the correctness of algorithm $\text{DPgoodmap}()$. Give a subset of nodes S' from some super-level of \hat{T}_1^f , let $\mathcal{F}_1(S')$ denote the forest consisting of all subtrees rooted at nodes in S' . For a node $w' \in T_2^g$, let $T_2(w')$ denote the subtree of T_2^g rooted at w' . We will now argue that $F(S, w) = 1$ if and only if there is a “partial” δ -good map from $\mathcal{F}_1(S) \rightarrow T_2(w)$.

More precisely: a continuous map $\alpha : \mathcal{F}_1(S) \rightarrow T_2(w)$ with (S, w) being valid is a *partial- ε -goodmap*, if properties (P1), (P2), and (P3) from Definition 5 hold (with T_1^f replaced by $\mathcal{F}_1(S)$ and T_2^g replaced by $T_2(w)$). Note that in the case of (P2), the condition in (P2) only

needs to hold for $u_1, u_2 \in \mathcal{F}_1(S)$ (implying that $\alpha(u_1), \alpha(u_2) \in T_2(w)$); that is, if $\alpha(u_1) \succeq \alpha(u_2)$ for $u_1, u_2 \in \mathcal{F}_1(S)$, then, we have $u_1^{2\varepsilon} \succeq u_2^{2\varepsilon}$. Note that while u_1 and u_2 are from $\mathcal{F}_1(S)$, $u_1^{2\varepsilon}$ and $u_2^{2\varepsilon}$ may not be in $\mathcal{F}_1(S)$ as it is possible that $f(u_1^{2\varepsilon}) = f(u_1) + 2\varepsilon \geq \text{height}(S)$. First, we observe the following:

▷ **Claim 11.** At the top level where $L_m^{(1)} = \{u = \text{root}(\widehat{T}_1^f)\}$ and $L_m^{(2)} = \{w = \text{root}(\widehat{T}_2^g)\}$ both contain only one node, if there is a partial- δ -good map from $\mathcal{F}_1(\{u\}) \rightarrow T_2(w)$, then there is a δ -good map from $|T_1^f| \rightarrow |T_2^g|$.

The correctness of our dynamic programming algorithm (part (ii) of Theorem 10) will follow from Claim 11 and Lemma 12 below. Lemma 12 is one of our key technical results, and its proof can be found in [25].

► **Lemma 12.** *For any valid pair (S, w) , $F(S, w) = 1$ if and only if there is a partial- δ -good map $\alpha : \mathcal{F}_1(S) \rightarrow T_2(w)$.*

Part (ii) of Theorem 10: time complexity

We now show that Algorithm `DPgoodmap()` can be implemented to run in the claimed time. Note that the augmented-tree nodes contain tree nodes of T_1^f and T_2^g , as well as the intersection points between tree arcs of T_1^f (resp. T_2^g) and super-levels. As there are at most $m = 2n$ number of super-levels in \mathcal{L}_1 and \mathcal{L}_2 , it follows that the total number of tree nodes in the augmented trees \widehat{T}_1^f and \widehat{T}_2^g is bounded by $O(nm) = O(n^2)$. In what follows, in order to distinguish between the tree nodes for the augmented trees (\widehat{T}_1^f and \widehat{T}_2^g) from the tree nodes of the original trees (T_1^f and T_2^g), we refer to nodes of the former as *augmented-tree nodes*, while the latter simply as *tree nodes*. It is important to note that the δ -degree-bound is defined with respect to the original tree nodes in T_1^f and T_2^g , not for the augmented trees (the one for the augmented trees can be significantly higher).

Our DP-algorithm essentially checks for the feasibility $F(S, w)$ of valid-pairs (S, w) . The following two lemmas bound the size of valid pairs, and their numbers. Their proofs are in [25].

► **Lemma 13.** *For any valid pair (S, w) , we have $|S| \leq \tau$ and $|\text{Ch}(S)| \leq \tau$, where $\tau = \tau_\delta(T_1^f, T_2^g)$ is the δ -degree-bound w.r.t. T_1^f and T_2^g .*

► **Lemma 14.** *Let $\tau = \tau_\delta(T_1^f, T_2^g)$ be the δ -degree-bound w.r.t. T_1^f and T_2^g . The total number of valid pairs that Algorithm `DPgoodmap`(T_1^f, T_2^g, δ) will inspect is bounded by $O(n^3 2^\tau)$, and they can be computed in the same time.*

To obtain the final time complexity for Algorithm `DPgoodmap`, consider computing $F(S, w)$ for a fixed valid pair (S, w) . This takes $O(1)$ time in the base case (the super-level index $i = 1$). Otherwise for the case $i > 1$, observe that $k = |\text{Ch}(w)| = \text{degree}(w) \leq \tau$, and $|\text{Ch}(S)| \leq \tau$ by Lemma 13. Hence the number of partitioning of $\text{Ch}(S)$ is bounded by $O(|\text{Ch}(S)|^k) = O(\tau^\tau)$. For each partition, checking conditions (F-1) and (F-2) takes $O(k)$ time; thus the total time needed to compute $F(S, w)$ is $O(k\tau^\tau) = O(\tau^{\tau+1})$. Combining this with Lemma 14, we have that the time complexity of Algorithm `DPgoodmap()` is bounded from above by $O(n^3 2^\tau \tau^{\tau+1})$, as claimed.

4.2 A Faster Algorithm

It turns out that we do not need to inspect all the $O(n^3 2^\tau)$ number of valid pairs as claimed in Lemma 14. We can consider only what we call sensible-pairs, which we define now.

► **Definition 15.** Given a valid-pair (S, w) , suppose S is from super-level $L_i^{(1)}$ and thus w is from super-level $L_i^{(2)}$. Then, (S, w) is a sensiblepair if either of the following two conditions hold:

- (C-1) S contains a tree node from $V(T_1^f)$, or its children $\text{Ch}(S) \subseteq L_{i-1}^{(1)}$ in the augmented tree \widehat{T}_1^f contains some tree node from $V(T_1^f)$, or the parents of nodes of S in the augmented tree \widehat{T}_1^f (which are necessarily from super-level $L_{i+1}^{(1)}$) contains some tree node from $V(T_1^f)$; or
- (C-2) w is a tree node of T_2^g , or $\text{Ch}(w) \subseteq L_{i-1}^{(2)}$ contains a tree node of T_2^g ; or the parent of w from super-level $L_{i+1}^{(2)}$ in the augmented tree \widehat{T}_2^g is a tree node of T_2^g .

Algorithm `DPgoodmap()` can be modified to Algorithm `modified-DP()` so that it only inspects sensible-pairs. The modification is non-trivial, and the reduction in the bound on number of sensible-pairs is by relating sensible-pairs to certain appropriately defined edge-list pairs $(A \subseteq E(T_1^f), \alpha \in E(T_2^g))$. The rather technical details can be found in [25]. We only summarize the main theorem below.

► **Theorem 16.**

- (i) Algorithm `modified-DP` (T_1^f, T_2^g, δ) returns “yes” if and only if $d_I(T_1^f, T_2^g) \leq \delta$.
- (ii) Algorithm `modified-DP` (T_1^f, T_2^g, δ) can be implemented to run in $O(n^2 2^\tau \tau^{\tau+2} \log n)$ time, where n is the total complexity of input trees T_1^f and T_2^g , and $\tau = \tau_\delta(T_1^f, T_2^g)$ is the δ -degree-bound w.r.t. T_1^f and T_2^g .

Note that if τ is constant, then the time complexity is $O(n^2 \log n)$.

5 Algorithms for Interleaving and Gromov-Hausdorff Distances

5.1 FPT Algorithm to Compute Interleaving Distance

In the previous section, we show how to solve the decision problem for interleaving distance between two merge trees T_1^f and T_2^g . We now show how to compute the interleaving distance δ^* , which is the smallest δ value such that $d_I(T_1^f, T_2^g) \leq \delta$ holds.

The main observation is that there exists a set Π of $O(n^2)$ number of candidate values such that δ^* is necessarily one of them. Specifically, let $\Pi_1 = \{|f(u) - g(w)| \mid u \in V(T_1^f), w \in V(T_2^g)\}$, $\Pi_2 = \{|f(u) - f(u')|/2 \mid u, u' \in V(T_1^f)\}$, and $\Pi_3 = \{|g(w) - g(w')|/2 \mid w, w' \in V(T_2^g)\}$. Set $\Pi = \Pi_1 \cup \Pi_2 \cup \Pi_3$. The proof of the following lemma can be found in [25].

► **Lemma 17.** The interleaving distance $\delta^* = d_I(T_1^f, T_2^g)$ satisfies that $\delta^* \in \Pi$.

Finally, compute and sort all candidate values in Π where by construction, $|\Pi| = O(n^2)$. Then, starting with δ being the smallest candidate value in Π , we perform algorithm `DPgoodmap` (T_1^f, T_2^g, δ) for each δ in Π in increasing order, till the first time the answer is “yes”. The corresponding δ value at the time is $d_I(T_1^f, T_2^g)$. Furthermore, note that for the degree-bound parameter, $\tau_\delta(T_1^f, T_2^g) \leq \tau_{\delta'}(T_1^f, T_2^g)$ for $\delta \leq \delta'$. Combining with Theorem 16, we can easily obtain the following trivial bound:

► **Theorem 18.** Let $\delta^* = d_I(T_1^f, T_2^g)$ and $\tau^* = \tau_{\delta^*}(T_1^f, T_2^g)$ be the degree-bound parameter of T_1^f and T_2^g w.r.t. δ^* . Then we can compute δ^* in $O(n^4 2^{\tau^*} (\tau^*)^{\tau^*+2} \log n)$ time.

However, it turns out that one can remove almost an $O(n^2)$ factor by using a double-binary search like procedure, as discovered by Kyle Fox. We include this improved result below and his argument below for completeness. See [25] for the proof.

► **Theorem 19.** Let $\delta^* = d_I(T_1^f, T_2^g)$ and $\tau^* = \tau_{\delta^*}(T_1^f, T_2^g)$ be the degree-bound parameter of T_1^f and T_2^g w.r.t. δ^* . Then we can compute δ^* in $O(n^2 2^{2\tau^*} (2\tau^*)^{2\tau^*+2} \log^3 n)$ time.

5.2 FPT-Algorithm for Gromov-Hausdorff Distance

Finally, we develop a FPT-algorithm to approximate the Gromov-Hausdorff distance between two input trees (T_1, d_1) and (T_2, d_2) . To approximate the Gromov-Hausdorff distance between two metric trees, we need to modify our parameter slightly (as there is no function defined on input trees any more). Specifically, now given a metric tree (T, d) , a ε -geodesic ball at $u \in |T|$ is simply $\widehat{B}_\varepsilon(u, T) = \{x \in |T| \mid d(x, u) \leq \varepsilon\}$.

Parameter τ : Given $\mathcal{T}_1 = (T_1, d_1)$ and $\mathcal{T}_2 = (T_2, d_2)$, define the ε -metric-degree-bound parameter $\widehat{\tau}_\varepsilon(\mathcal{T}_1, \mathcal{T}_2)$ to be the largest sum of degrees of all tree nodes within any ε -geodesic ball in T_1 (w.r.t. metric d_1) or in T_2 (w.r.t. d_2).

We obtain our main result for approximating the Gromov-Hausdorff distance between two metric trees within a factor of 14. We note that to obtain this result, we need to also relate the ε -metric-degree-bound parameter for metric trees with the ε -degree-bound parameter used for interleaving distance for the special geodesic functions we use (in fact, we will show that $\widehat{\tau}_\delta \leq \tau_\delta \leq \widehat{\tau}_{2\delta}$). The proof of the following main theorem of this section can be found [25].

► **Theorem 20.** *Given two metric trees $\mathcal{T}_1 = (T_1, d_1)$ and $\mathcal{T}_2 = (T_2, d_2)$ where the total number of vertices of T_1 and T_2 is n , we can 14-approximate the Gromov-Hausdorff distance $\widehat{\delta}^* = \delta_{GH}(\mathcal{T}_1, \mathcal{T}_2)$ in $O(n^4 \log n + n^2 2^{\widehat{\tau}+2} \log^3 n)$ time, where $\widehat{\tau} = 2\widehat{\tau}_{28\widehat{\delta}^*}(\mathcal{T}_1, \mathcal{T}_2)$ is twice the metric-degree-bound parameter w.r.t. $28\widehat{\delta}^*$.*

Remarks

We remark that the time complexity of the FPT approximation algorithm of [22] contains terms n^k , where k is the parameter and could be large in general – Indeed, k is the cardinality of an ε -net of one of the input metric spaces, and ε also appears as an additive approximation term for algorithm. In contrast, the dependency of our algorithm on the parameter $\widehat{\tau}$ is roughly $O(2^{O(\widehat{\tau})})$, and our algorithm has only constant multiplicative approximation factor. On the other hand, note that the algorithm of [22] works for general finite metric spaces. We also remark that the Gromov-Hausdorff distance between two metric spaces (X, d_X) and (Y, d_Y) measures their *additive distortion*, and thus is not invariant under scaling. In particular, suppose the input two metric spaces $\mathcal{T}_1 = (T_1, d_1)$, $\mathcal{T}_2 = (T_2, d_2)$ scale by the same amount to a new pair of input trees $\mathcal{T}'_1 = (T'_1, d'_1 = c \cdot d_1)$, $\mathcal{T}'_2 = (T'_2, d'_2 = c \cdot d_2)$. Then the new Gromov-Hausdorff distance between them $\delta_{GH}(\mathcal{T}'_1, \mathcal{T}'_2) = c \cdot \delta_{GH}(\mathcal{T}_1, \mathcal{T}_2)$. However, note that the metric-degree-bound parameter for the new trees satisfies $\widehat{\tau}_{c\delta}(\mathcal{T}'_1, \mathcal{T}'_2) = \widehat{\tau}_\delta(\mathcal{T}_1, \mathcal{T}_2)$. Hence the time complexity of our algorithm to approximate the Gromov-Hausdorff distance $\delta_{GH}(\mathcal{T}'_1, \mathcal{T}'_2)$ for scaled metric-trees \mathcal{T}'_1 and \mathcal{T}'_2 **remains the same** as that for approximating the Gromov-Hausdorff distance $\delta_{GH}(\mathcal{T}_1, \mathcal{T}_2)$.

6 Concluding Remarks

In this paper, by re-formulating the interleaving distance, we developed the first FPT algorithm to compute the interleaving distance *exactly* for two merge trees, which in turn leads to an FPT algorithm to approximate the Gromov-Hausdorff distance between two metric trees.

We remark that the connection between the Gromov-Hausdorff distance and the interleaving distance is essential, as the interleaving distance has more structure behind it, as well as certain “order” (along the function associated to the merge tree), which helps to develop dynamic-programming type of approach. For more general metric graphs (which

represent much more general metric spaces than trees), it would be interesting to see whether there is a similar relation between the Gromov-Hausdorff distance of metric graphs and the interleaving distance between the so-called Reeb graphs (generalization of merge trees).

References

- 1 Pankaj K. Agarwal, Kyle Fox, Abhinandan Nath, Anastasios Sidiropoulos, and Yusu Wang. Computing the Gromov-Hausdorff Distance for Metric Trees. *ACM Trans. Algorithms*, 14(2):24:1–24:20, 2018.
- 2 Richa Agarwala, Vineet Bafna, Martin Farach, Mike Paterson, and Mikkel Thorup. On the approximability of numerical taxonomy (fitting distances by tree metrics). *SIAM Journal on Computing*, 28(3):1073–1085, 1998.
- 3 Nir Ailon and Moses Charikar. Fitting tree metrics: Hierarchical clustering and phylogeny. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 73–82. IEEE, 2005.
- 4 Noga Alon, Mihai Bădoiu, Erik D Demaine, Martin Farach-Colton, MohammadTaghi Hajiaghayi, and Anastasios Sidiropoulos. Ordinal embeddings of minimum relaxation: general properties, trees, and ultrametrics. *ACM Transactions on Algorithms (TALG)*, 4(4):46, 2008.
- 5 Ulrich Bauer, Xiaoyin Ge, and Yusu Wang. Measuring Distance between Reeb Graphs. In *30th Annual Sympos. on Comput. Geom.*, page 464, 2014.
- 6 Ulrich Bauer, Claudia Landi, and Facundo Mémoli. The Reeb Graph Edit Distance is Universal. *CoRR*, abs/1801.01866, 2018. [arXiv:1801.01866](https://arxiv.org/abs/1801.01866).
- 7 Silvia Biasotti, Daniela Giorgi, Michela Spagnuolo, and Bianca Falcidieno. Reeb graphs for shape analysis and applications. *Theor. Comput. Sci.*, 392(1-3):5–22, 2008.
- 8 Philip Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3):217–239, June 2005.
- 9 Alexander M. Bronstein, Michael M. Bronstein, and Ron Kimmel. Efficient computation of isometry-invariant distances between surfaces. *SIAM J. on Sci. Comput.*, 28(5):1812–1836, 2006.
- 10 Mihai Bădoiu, Piotr Indyk, and Anastasios Sidiropoulos. Approximation algorithms for embedding general metrics into trees. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 512–521. Society for Industrial and Applied Mathematics, 2007.
- 11 Victor Chepoi, Feodor F Dragan, Ilan Newman, Yuri Rabinovich, and Yann Vaxes. Constant approximation algorithms for embedding graph metrics into trees and outerplanar graphs. *Discrete & Computational Geometry*, 47(1):187–214, 2012.
- 12 Vin de Silva, Elizabeth Munch, and Amit Patel. Categorized Reeb Graphs. *Discrete & Computational Geometry*, 55(4):854–906, June 2016.
- 13 Michael Fellows, Fedor Fomin, Daniel Lokshtanov, Elena Losievskaja, Frances A Rosamond, and Saket Saurabh. Parameterized Low-distortion Embeddings-Graph metrics into lines and trees. *arXiv preprint*, 2008. [arXiv:0804.3028](https://arxiv.org/abs/0804.3028).
- 14 Xiaoyin Ge, Issam Safa, Mikhail Belkin, and Yusu Wang. Data Skeletonization via Reeb Graphs. In *Proc. 25th Annu. Conf. Neural Information Processing Systems (NIPS)*, pages 837–845, 2011.
- 15 Mikhail Gromov. *Metric Structures for Riemannian and Non-Riemannian Spaces*. Birkhäuser Basel, 2007.
- 16 Alexander Hall and Christos Papadimitriou. Approximating the Distortion. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, volume 3624 of *Lecture Notes in Computer Science*, pages 111–122. Springer Berlin Heidelberg, 2005.
- 17 Franck Hétyroy and Dominique Attali. Topological quadrangulations of closed triangulated surfaces using the Reeb graph. *Graph. Models*, 65(1-3):131–148, 2003.
- 18 Tao Jiang, Lusheng Wang, and Kaizhong Zhang. Alignment of Trees - An Alternative to Tree Edit. *Theor. Comput. Sci.*, 143(1):137–148, 1995. doi:10.1016/0304-3975(95)80029-9.

- 19 Facundo Mémoli and Guillermo Sapiro. A Theoretical and Computational Framework for Isometry Invariant Recognition of Point Cloud Data. *Found. of Comput. Math.*, 5(3):313–347, 2005.
- 20 Dmitriy Morozov, Kenes Beketayev, and Gunther H. Weber. Interleaving Distance between Merge Trees. In *Workshop on Topological Methods in Data Analysis and Visualization: Theory, Algorithms and Applications*, 2013.
- 21 Facundo Mémoli. Some Properties of Gromov–Hausdorff Distances. *Discrete & Computational Geometry*, pages 1–25, 2012. 10.1007/s00454-012-9406-8. doi:10.1007/s00454-012-9406-8.
- 22 Felix Schmiedl. Computational Aspects of the Gromov–Hausdorff Distance and its Application in Non-rigid Shape Matching. *Discrete & Computational Geometry*, 57(4):854–880, June 2017. doi:10.1007/s00454-017-9889-4.
- 23 A. Sidiropoulos, D. Wang, and Y. Wang. Metric embeddings with outliers. In *Proc. 28th ACM-SIAM Sympos. Discrete Algorithms (SoDA)*, pages 670–689, 2017.
- 24 Julien Tierny. *Reeb graph based 3D shape modeling and applications*. PhD thesis, Université des Sciences et Technologies de Lille, 2008.
- 25 Elena Farahbakhsh Touli and Yusu Wang. FPT-algorithms for computing Gromov-Hausdorff and interleaving distances between trees. *CoRR*, abs/1811.02425, 2018. arXiv:1811.02425.
- 26 Kaizhong Zhang and Tao Jiang. Some MAX SNP-hard results concerning unordered labeled trees. *Information Processing Letters*, 49(5):249–254, 1994.